„Compositional Modeling“ Ansatz zur Benutzerschnittstellengenerierung am Beispiel telemedizinischer Anwendungen

Dissertation

Zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik
von

Thomas Königsmann
Dortmund

2011

Tag der mündlichen Prüfung: 20.05.2011

Dekan: Prof. Dr. Peter Buchholz

Gutachter: Prof. Dr. Jakob Rehof
Prof. Dr. Heinrich Müller

Abstract

In den letzten fünf Jahren hat die Entwicklung neuer Interaktionsformen und damit einhergehend auch die Vielfalt von Endgeräten in erheblichem Maße zugenommen. Die Multi-Touch-Steuerung des iPhones von Apple, der Bewegungssensor der Nintendo Wii und auch E-Book-Reader (bspw. das Kindl von Amazon) erschließen neue Kundenkreise. Sie sind auch Enabler für neue Geschäftsfelder, wie beispielsweise telemedizinische Dienstleistungen und das Ambient Assisted Living (kurz AAL) im häuslichen Umfeld.

Die Bereitstellung von IT-Diensten in solchen Geschäftsfeldern erfordert die verstärkte Betrachtung der Benutzerschnittstelle. Begründet ist diese Forderung nicht nur durch unterschiedliche Endgeräte, sondern durch die Notwendigkeit eine Anpassung von Benutzerschnittstellen auf bestimmte Zielgruppen (beispielsweise ältere Menschen im AAL) durchführen zu können. Die Portierung von Benutzerschnittstellen auf unterschiedliche Endgeräte und die Berücksichtigung der Bedürfnisse von bestimmten Nutzergruppen stellen zunehmend ein Problem dar, für das Lösungen gesucht werden.

Vor diesem Hintergrund wird ein Verfahren entwickelt, das automatisiert Benutzerschnittstellen durch Komposition von Benutzerschnittstellenbausteinen generiert. Als Leitmotiv für die Generierung von Benutzerschnittstellen wurde der „Compositional Modeling“-Ansatz von Falkenhainer und Forbus gewählt. Im Kern beschreibt das „Compositional Modeling“ eine wissensbasierte Inferenzmaschine, die basierend auf domänenspezifischen Wissensbausteinen in der Lage ist, durch Komposition ein Modell aufzubauen, das auf eine gegebene Fragestellung hin optimiert ist.

In der vorliegenden Arbeit erfolgt die Übertragung dieses Ansatzes auf die Benutzerschnittstellengenerierung. Dies erfordert die Konzeption und Bereitstellung von Modellen, Wissensbasen und Algorithmen für die Generierung. Benötigte Modelle sind:

- Eine *Interaktionsbeschreibung*, die es ermöglicht zu spezifizieren, wie die Interaktionen mit der Benutzerschnittstelle erfolgen sollen, ohne Aussagen zu treffen, wie diese zu realisieren sind.
- Ein *Nutzungskontextmodell*, mit dem Anforderungen (Endgerät, Nutzer, Umgebung der Nutzung) beschrieben werden, welche die zu generierende Benutzerschnittstelle erfüllen muss.
- *Benutzerschnittstellenbausteine*, die wiederverwendbare Teile eines Interaktionsablaufes spezifizieren und aus denen Benutzerschnittstellen generiert werden können.

Basierend auf diesen Modellen wird ein Verfahren erarbeitet, das in der Lage ist, aus einer Interaktionsbeschreibung, einem Nutzungskontext und einer gegebenen Wissensbasis (bestehend aus Benutzerschnittstellenbausteinen) eine ausführbare Benutzerschnittstelle zu generieren.

Veranschaulicht werden die Modelle, Wissensbasen und Verfahren an dem telemedizinischen Anwendungsfall „Adipositas-Begleiter“. Anhand dieses erfolgt eine Evaluation des Verfahrens, indem für diese Anwendungsdomäne Wissensbasen aufgebaut werden und damit schließlich Benutzerschnittstellen für drei Nutzungskontexte generiert werden, die sich jeweils durch unterschiedliche Endgeräte (PC, Smart-Phone, TV), Nutzer (Youngster, Mid Ager, Best Ager) und Umgebungen der Nutzung (unterwegs, zuhause, bei der Arbeit) unterscheiden.

Abbildungsverzeichnis

ABBILDUNGSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

Tabellenverzeichnis

1 Einleitung

Die fortschreitende Nutzung von IT-Technologien im alltäglichen Leben ist eine der wesentlichen Entwicklungen der letzten Jahre. Die Verbreitung des Internets ist dabei zweifelsfrei ein entscheidender Schritt in diese Richtung gewesen. Die globale Verfügbarkeit riesiger Informationsmengen, die Bereitstellung neuer Kommunikationsmedien und Dienstleistungen, die von jedem Heimcomputer aus verfügbar sind, verändern die Sichtbarkeit und den Umgang mit IT-Technologien. Heutzutage ist deren Nutzung tief im Alltag verwurzelt. Beginnend mit dem Nachrichtenaustausch per E-Mail, über die Informationssuche, Online-Shopping, Online-Banking bis hin zur Internet-Telefonie oder auch Video-On-Demand-Diensten ist IT zu einem festen Bestandteil des alltäglichen Lebens geworden (vgl. TNS Infratest Studie in [TNS09]).

Die Verbreitung von Internet-fähigen Mobiltelefonen, PDAs (Personal Digital Assistant), intelligenten mobilen Navigationssystemen, aber auch Set-Top-Boxen, die dem heimischen Fernseher den Internet-Zugang ermöglichen, bietet Menschen einen natürlicheren Zugang zur IT als über den herkömmlichen Desktop-PC (vgl. [BIT09]). Der Erfolg von Apples iPhone oder Nintendos Wii veranschaulicht, dass völlig neue Zielgruppen erreicht werden können, wenn sich Anwendungen an den Bedürfnissen und Fähigkeiten ihrer Nutzer orientieren. Die aktuell von der EU und auch vom BMBF ausgeschriebenen Projekte im Bereich von AAL (Ambient Assisted Living) verdeutlichen weiterhin, wie groß der Bedarf ist, beispielsweise älteren Menschen den Zugang zu IT-Diensten zu erschließen (vgl. [GC10]). Der dabei gewählte „ambiente“ Ansatz veranschaulicht, dass auch hier auf eine Entwicklung weg vom herkömmlichen Desktop-PC hin zu in das alltägliche Umfeld integrierten und auf den Nutzer zugeschnittenen Benutzerschnittstellen gewirkt wird.

Dabei ist es bei weitem nicht mehr ausreichend, Anwendungen auf unterschiedliche Endgeräte zu portieren und damit unterschiedliche Softwareplattformen und Display-Größen zu bedienen. Endgeräte unterscheiden sich heutzutage in einem viel größerem Umfang. Es stehen neue Interaktionsformen zur Verfügung, wie es der Nintendo Wii Controller bietet, der Beschleunigungen registriert. Der gestenorientierte Ansatz von Multi-Touch-Displays, wie es das Apple iPhone ermöglicht, aber auch gänzlich neue Technologien wie „Tangitable User Interfaces“, welche reale Objekte in die Interaktion mit einbeziehen (vgl. [DKSS09]), sind weitere Beispiele für neue Interaktionsformen. Solche Technologien eröffnen völlig neue Möglichkeiten mit einer Anwendung zu interagieren.

Zunehmend wird aber erkannt, dass die Endgeräteproblematik nur einen Teil der eigentlichen Herausforderung darstellt. Anwendungen müssen sich an den Bedürfnissen und Anforderungen des Menschen orientieren. Das bedeutet aber, dass nicht nur die physischen Fähigkeiten im Umgang mit Tastatur, Maus oder neueren Schnittstellen wie Controller, Sprache oder Gestensteuerung zu betrachten sind, sondern auch die mentalen Fähigkeiten des Menschen, Abläufe in Benutzerschnittstellen zu verstehen und zielgerichtet einzusetzen, berücksichtigt werden müssen.

Der vorliegenden Arbeit liegt die Vision zu Grunde „kontextorientierte Benutzerschnittstellen“ anbieten zu können. Kontextorientiert bedeutet in dieser Arbeit, dass sich die Benutzerschnittstelle an die Bedürfnisse und Fähigkeiten des Benutzers, an das verwendete Endgerät und auch an die Umgebung der Nutzung anpasst. Ziel ist es, Hürden und Hemmschwellen für die Nutzung von Informations- und Kommunikationstechnologien abzubauen.

1.1 Problembeschreibung

Die Vision „kontextorientierte Benutzerschnittstellen“ anbieten zu können, erfordert bei der Realisierung ein Höchstmaß an Flexibilität. Ein wesentlicher Faktor sind dabei die Anforderungen, die an eine „kontextorientierte Benutzerschnittstelle“ zu stellen sind. Um einen ersten Einblick in die Komplexität zu gewähren, wird exemplarisch ein auf die drei Kategorien Endgerät, Nutzer und Umgebung der Nutzung ausgerichteter Anforderungskatalog vorgestellt, wobei jede Kategorie durch drei Parameter definiert wird:

- Endgerät
  - Interaktionsmedien: Zur Verfügung stehende Interaktionsmedien wie beispielsweise Maus, Tastatur, Controller, Sprachsteuerung, usw.
  - Interaktionsobjekte: Zur Verfügung stehende Interaktionsobjekte wie beispielsweise Fenster, Buttons, Widgets, Listen, Menüs, usw.
  - Ausgabemedien: Zur Verfügung stehende Ausgabemedien wie Display, Audio, Sprachausgabe, usw.
- Nutzer
  - Physische Fähigkeiten: Fähigkeit zur Hand-Augen-Koordination, Feinmotorik, Fähigkeit zur Artikulation (Sprachsteuerung), usw.
  - Mentale Fähigkeiten: Fähigkeiten zur Konzentration, Begreifen von parallelen Abläufen in Mehrfenstersystemen, usw.
  - Kenntnisse von Bedienkonzepten: Browser-basierte Anwendungen, Umgang mit Touch-Displays, „Wizzards“ für geführte Interaktionen, usw.
- Umgebung der Nutzung
  - Physische Umgebung: Umgebungslärm, Lichtverhältnisse, usw.
  - Einbettung in die Umgebung: Möglichkeit zur Konzentration auf die Anwendung (z.B. sehr gering als Autofahrer), usw.
  - Personen in der direkten Umgebung: Persönliche Inhalte müssen verborgen bleiben, Passwörter müssen geschützt werden, usw.

Diese Darstellung stellt nur einen sehr kleinen Ausschnitt möglicher Anforderungen dar, die für die Realisierung von Benutzerschnittstellen von Bedeutung sein können.

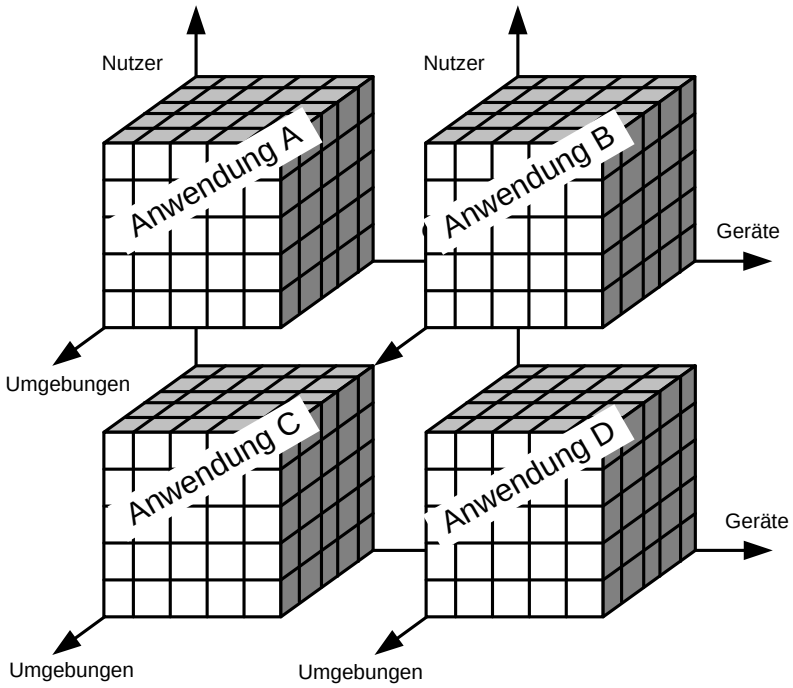


Abbildung 1: Komplexität „kontextorientierter Benutzerschnittstellen“

Abbildung 1 visualisiert vereinfacht die Menge an unterschiedlichen Benutzerschnittstellen, die benötigt werden, um den Anforderungen aus den drei Kategorien „Nutzer“, „Gerät“ und „Umgebung“ gerecht zu werden. Dabei wird ein Koordinatensystem verwendet, um Anforderungen an eine Benutzerschnittstelle zu kennzeichnen. Die X-Achse wird für die Darstellung von Anforderungen verwendet, die sich aus der Nutzung einer Endgerätekonfiguration (bestehend aus Interaktionsmedien, Interaktionsobjekten und Ausgabemedien) ergeben. Jeder Punkt auf der X-Achse repräsentiert eine Gerätekonfiguration. Analog dazu steht die Y-Achse für unterschiedliche Benutzertypen und die Z-Achse für die Umgebung der Nutzung.

Reduziert man die Anforderungen auf ein solches diskretes Koordinatensystem, so müssen geeignete Benutzerschnittstellen für jeden Punkt im Koordinatensystem entwickelt werden. In Abbildung 1 wurden exemplarisch auf jeder Achse fünf Punkte herausgegriffen, um die Fülle (5x5x5=125) unterschiedlicher Kombinationen und damit auch Benutzerschnittstellen aufzuzeigen. Auch wenn Teile der Anforderungen zusammengefasst werden können, bzw. aufgrund von widersprüchlichen Anforderungen nicht realisierbar sind, bleibt dennoch eine erhebliche Anzahl an Kombinationen, die betrachtet werden müssen, um „kontextorientierte Benutzerschnittstellen“ anbieten zu können.

Dabei ist zu berücksichtigen, dass die Realisierung jeder einzelnen Benutzerschnittstelle für jede Anwendung separat betrieben werden muss, wie durch die vier Koordinatensysteme (Anwendung A, B, C und D) in Abbildung 1 dargestellt wird. Es ist somit leicht nachvollziehbar, dass eine Implementierung dieser unterschiedlichen Benutzerschnittstellen bereits für eine kleine Anzahl an möglichen Endgeräte-, Benutzer- und Umgebungs-Konfigurationen nicht mehr realistisch zu handhaben ist.

Wenn „kontextorientierte Benutzerschnittstellen“ Realität werden sollen, besteht die Notwendigkeit, Modelle, Werkzeuge und Methoden bereit zu stellen, mit denen sich die Generierung von Benutzerschnittstellen automatisieren lässt. Betrachtet man die exemplarisch eingeführten Anforderungen, die sich aus der Betrachtung des Nutzers, des Endgerätes und der Umgebung der Nutzung ergeben, so ist zu berücksichtigen, dass solch eine abstrakte Betrachtung der Anforderungen zwar ausreichend für die Einführung der Problemstellung der Benutzerschnittstellengenerierung ist, aber zu wenig Input für einen automatisierten Generierungsprozess liefern kann. Die Interpretation dieser Daten ist bereits für einen Menschen schwierig zu handhaben. Ein Entwickler muss entscheiden, welche dieser Anforderungen relevant sind, wie mit Widersprüchen zu verfahren ist, und schließlich bewerten, welche Auswirkungen diese Anforderungen auf die Gestaltung und die Abläufe in der Benutzerschnittstelle haben.

Ein Leitbild für die automatisierte Generierung von Software-Systemen bieten Service-orientierte Ansätze (vgl. [BHS08]). Technologisch ist insbesondere der Themenbereich der Service-Orchestrierung (vgl. [Pel03]) von Bedeutung, der weit über die Bereitstellung einer Dienstelandschaft hinausgeht und bereits auf die Unterstützung von kompletten Geschäftsprozessen durch adaptive Workflows oder auf die semantische Komposition von Diensten abzielt. Diesen Arbeiten liegt die Vision zu Grunde, aus Geschäftsprozessen Dienste zu identifizieren, zu komponieren und schließlich ausführbaren Code zu generieren. Dienste repräsentieren dabei Software-Bausteine, die in unterschiedlichen Anwendungen wieder verwendet werden können. Insbesondere der Service-Komposition (vgl. [Pel03]) als Schlüsseltechnologie wird zugesprochen, dass durch sie die Effizienz der Anwendungsgenerierung und auch die Flexibilität von Anwendungen optimiert werden kann.

Es liegt nahe, eine ähnliche Vision für die Benutzerschnittstellengenerierung zu entwickeln. Auch sie muss Teil eines Software-Entwicklungsprozesses sein und ist prädestiniert für bausteinorientierte Konzepte. Softwarebausteine, die Benutzerschnittstellen realisieren, kapseln ein Höchstmaß an Wissen, das sich nur schwer formalisieren lässt. Gelingt es, solche Bausteine zu identifizieren und für eine Generierung von Benutzerschnittstellen nutzbar zu machen, so können Werkzeuge definiert werden, die Lösungen für unterschiedliche Anforderungsprofile liefern können. Dieses Vorgehen steht aber in Diskrepanz zum verfügbaren Stand der Technik (vgl. [FHSS09]). Obwohl die aktuell diskutierten Ansätze im Umfeld der Service-orientierten Software-Entwicklung auf bausteinorientierten Konzepten begründet sind, erfolgt die Betrachtung der Benutzerschnittstelle nur sporadisch. Und auch weiterführende Konzepte wie beispielsweise SaaS (Software as a Service) (vgl. [BHS08]) klammern die Benutzerschnittstelle weitgehend aus und thematisieren bereits die Ebene der Geschäftsmodellentwicklung.

Aufgabe dieser Arbeit ist es, Verfahren und Methoden für die bausteinorientierte Benutzerschnittstellengenerierung zu konzipieren und zu evaluieren. Hierdurch soll einerseits die Vision der „kontextorientierten Benutzerschnittstellen“ weiter entwickelt und andererseits Entwicklungsarbeiten im Bereich der bausteinorientierten Benutzerschnittstellengenerierung geleistet werden. Wie bereits am Beispiel der Problemstellung zur Definition von Anforderungen an eine Benutzerschnittstelle diskutiert wurde, sind

Einschränkungen und Konkretisierungen notwendig, wenn der Schritt von einer konzeptionellen Lösung hin zu praktisch nutzbaren Methoden und Werkzeugen erfolgen soll.

Im Rahmen dieser Arbeit werden diese notwendigen Restriktionen und Konkretisierungen durch die Fokussierung auf eine Anwendungsdomäne erreicht.

Als Anwendungsdomäne wurde die Telemedizin gewählt, da dieser Bereich in einem Höchstmaß auf „kontextorientierte Benutzerschnittstellen“ angewiesen ist. Begründet ist dies durch höchst heterogene Zielgruppen (junge Menschen in der Prävention, aber auch ältere Menschen und Menschen mit körperlichen und/oder mentalen Einschränkungen), die meist nicht über herkömmliche Desktop-PCs erreicht werden können. Weiterhin sind in diesem Bereich IT-Dienste erforderlich, die im Sinne des Ambient Assisted Livings (kurz AAL, vgl. [GC10]) in den Alltag integriert sind. Solche IT-Dienste werden in unterschiedlichen Umgebungen genutzt und müssen über unterschiedliche Endgeräte erreichbar sein.

1.2 Ansatz und Ziele der Arbeit

Die der Arbeit zugrunde liegende Aufgabenstellung greift den Kompositionsgedanken auf, wie er in der Service-orientierten Softwareentwicklung zu finden ist, und überträgt diesen auf die Benutzerschnittstellengenerierung. Als Grundgedanke steht dabei die These im Vordergrund, dass die Implementierung von Benutzerschnittstellen ein Höchstmaß an Wissen kapselt, das sich nur schwer formalisieren lässt. Folglich liegt die Zielsetzung der Arbeit darin, Ansätze zu finden, mit denen Benutzerschnittstellen in wieder verwendbare Bausteine zerlegt werden können. Für diese Bausteine sind Methoden und Konzepte zu erarbeiten, aus denen schließlich neue Benutzerschnittstellen generiert werden.

Als Leitmotiv für die Generierung von Benutzerschnittstellen wurde der „Compositional Modeling“-Ansatz gewählt, der erstmals von Falkenhainer und Forbus in [FF91] eingeführt wurde. Im Kern beschreibt das „Compositional Modeling“ eine wissensbasierte Interferenzmaschine, die basierend auf domänenspezifischen Wissensbausteinen (Modellfragmenten) in der Lage ist, ein Szenario (in [FF91] Scenario Model genannt) aufzubauen, das auf eine gegebene Fragestellung (in [FF91] Query genannt) hin optimiert ist. Das „Compositional Modeling“ repräsentiert einen Ansatz, der die Komposition für die Modellgenerierung nutzt. Als besondere Stärke dieses Vorgehens gilt dabei, dass aus einer relativ kleinen Wissensbasis eine große Menge an unterschiedlichen Modellen generiert werden kann (vgl. [FSZ07]).

Falkenhainer und Forbus nutzen die folgende Fragestellung, um den Ansatz des „Compositional Modeling“ zu veranschaulichen:

Gegeben sei:

- eine **Szenariobeschreibung**, welche die Struktur (verwendete Objekte) und Aussagen über ihr Verhalten beinhaltet;
- eine **Domain Theory**, welche aus einer Menge von Modellfragmenten (in [FF91] Domänenfragmente genannt) und Regeln über ihren Einsatz besteht;

- eine **Query**, die eine Fragestellung definiert, die durch das Scenario Model beantwortet werden soll.

Zu produzieren ist:

- ein **Scenario Model** (bestehend aus Modellfragmenten und einer Abbildung dieser auf die Szenariobeschreibung), das zu der gegebenen Query möglichst nützlich (beispielsweise definiert durch geringe Komplexität) und kohärent ist. Das Scenario Model ist das Ergebnis der Modellgenerierung, und definiert eine Instanz der Szenariobeschreibung, die sich durch die Wahl der Modell-Fragmente der Domain Theory auszeichnet.

Ziel ist es, diese Fragestellung für die Generierung von Benutzerschnittstellen zu nutzen. In Analogie wird als Kernthese der Arbeit formuliert:

> *Benutzerschnittstellen können automatisiert aus Modellfragmenten durch Komposition generiert werden, wenn man in der Lage ist, analog zum „Compositional Modeling" eine Szenariobeschreibung, eine Domain Theory und eine Query zu definieren.*

Diese Kernthese impliziert in einem ersten Schritt die Notwendigkeit, Modelle zu entwickeln, die im Hinblick auf die Zielsetzung der Benutzerschnittstellengenerierung geeignet erscheinen. Dabei ist anzunehmen, dass das Scenario Model als Zielmodell der Generierung durch eine ausführbare Benutzerschnittstelle repräsentiert wird. Die drei übrigen Modelle werden (als Quellenmodelle der Generierung) im Rahmen dieser Arbeit ausgerichtet auf die Benutzerschnittstellengenerierung erarbeitet:

- Eine **Szenariobeschreibung** wird durch ein Modell (im Folgenden Interaktionsbeschreibung genannt) definiert, welches auf einer abstrakten Ebene die Interaktionen des Benutzers mit dem System beschreibt.
- Die **Domain Theory** besteht aus einer Menge von Modellfragmenten. Modellfragmente repräsentieren Spezifikationen, die Benutzerschnittstellen auf der Basis von Masken, Buttons und Eingabefeldern beschreiben. Modellfragmente in der Domain Theory können in dem Sinne redundant sein, dass unterschiedliche Modellfragmente den „gleichen Teil" einer Benutzerschnittstelle auf unterschiedliche Weise realisieren.
- Die **Query** kann durch ein spezialisiertes Kontextmodell (im Folgenden Nutzungskontext genannt) beschrieben werden, das Nutzerpräferenzen, Endgeräterestriktionen, aber auch das Umfeld (in der Wohnung, in der Bahn, Lichtverhältnisse) modelliert.

Neben der Definition von Modellierungsmethodiken für diese drei Modelle stellt der Generierungsprozess selbst eine der wesentlichen Leistungen dieser Arbeit dar. Insbesondere hervorzuheben ist dabei die entwickelte Methodik zur automatischen Komposition von Benutzerschnittstellenfragmenten im Rahmen der Invarianten, die sich aus der Interaktionsbeschreibung ableiten lassen.

Am Beispiel einer konkreten Anwendung in der Anwendungsdomäne Telemedizin wird ein entsprechendes Wissensrepository exemplarisch aufgebaut, die Modelle instanziert und schließlich aus diesen Modellen Benutzerschnittstellen generiert, um damit die Kernthese der Arbeit im Sinne eines „Proof of Concept“ und einer Evaluierung zu belegen.

1.3 Leistungen der Arbeit in dem Themenfeld Benutzerschnittstellengenerierung

Benutzerschnittstellen und die Benutzerschnittstellengenerierung stellen in der Informatik ein weit gefächertes Forschungsfeld dar, das sehr unterschiedliche Bereiche umfasst. Der Themenumfang reicht von der psychologischen Betrachtung, wie Anwender mit Benutzerschnittstellen interagieren und gestalterischen Fragestellung über das User-Interface Design, über Usibility- und Performance-Betrachtungen bis hin zu formalen Beschreibungssprachen und wissensbasierten Inferenzmaschinen für Benutzerschnittstellen. Diese Forschungsschwerpunkte stellen wichtige und notwendige Themenbereiche dar, die für eine ganzheitliche Betrachtung dieses hoch komplexen Forschungsfeldes notwendig sind. Im Rahmen dieser Arbeit kann das Forschungsfeld nicht in Gänze betrachtet werden, so dass an dieser Stelle definiert wird, welche Beiträge geleistet werden und wo noch zukünftige Arbeiten folgen müssen.

Thematisch ordnet sich der „Compositional Modeling“-Ansatz der „modellbasierten Entwicklung von Benutzerschnittstellen“ unter, wie sie in [Van08] definiert wurde. Im Rahmen dieser Arbeit werden Modelle und Spezifikationen entwickelt, welche die Ausgangsbasis für einen Generierungsprozess bilden. Im Einzelnen sind dies:

- Weiterentwicklung eines Interaktionsmodells für die Interaktionsbeschreibung. Es werden existierende Interaktionsmodelle und Spezifikationstechniken aufgenommen und im Rahmen dieser Arbeit für die Generierung von „kontextorientierten Benutzerschnittstellen“ weiterentwickelt. Um eine notwendige Abstraktionsebene zu erreichen, wird erstmals versucht, Metaphern in Benutzerschnittstellen als Leitbild für die Interaktionsmodellierung zu verwenden.
- Entwicklung eines Kontextmodells für die Beschreibung von Anforderungen an die Benutzerschnittstelle. Ausgangslage sind auch hier in der Literatur und Praxis etablierte Kontextmodelle, die auf das Themenfeld Benutzerschnittstellen ausgerichtet sind und schließlich soweit konkretisiert werden, dass sie für eine Generierung von Benutzerschnittstellen nutzbar sind.
- Definition von wieder verwendbaren Bausteinen für die Benutzerschnittstellengenerierung. Für die Beschreibung der Bausteine wird dabei wieder auf existierende Arbeiten zurückgegriffen, indem etablierte XML-Beschreibungssprachen verwendet werden. Erweiterungen dieser Beschreibungssprachen sind nicht notwendig und werden nicht thematisiert.

Neben der Spezifikation und Weiterentwicklung von Modellen für modellbasierte Benutzerschnittstellen wird mit dem „Compositional Modeling“ ein neues Vorgehensmodell für die Benutzerschnittstellengenerierung eingeführt. Im Gegensatz zu bisherigen Lösungen wird ein bausteinorientierter Ansatz verfolgt,

der als Methode für die Generierung nicht die kontinuierliche Detaillierung von Modellen und Modelltransformationen nutzt, sondern durch Komposition von existierenden Bausteinen neue Benutzerschnittstellen realisiert. Dem Ansatz folgend müssen neue Methoden und Verfahren, Bibliotheken und Wissensbasen aufgebaut werden, die in dieser Form neu für den Bereich der „modellbasierten Entwicklung von Benutzerschnittstellen“ sind.

Die Fokussierung auf den Bereich der „patientenorientierten telemedizinischen Dienste“ liefert weiterhin Input aus einem spannenden Anwendungsfeld. Im Rahmen der Arbeit kann aber kein vollständiges Modell der Domäne für die Benutzerschnittstellengenerierung erarbeitet werden. Nur punktuell werden relevante Aspekte aus der Domäne herausgegriffen. Vorbild ist dabei das konkrete Projekt „Adipositas-Begleiter“, bei dem stark übergewichtige Menschen nach einer stationären Therapie in ihrem häuslichen Umfeld durch telemedizinische Dienste betreut werden. Aus diesem Projekt wird domänenspezifisches Wissen extrahiert und sowohl als Rahmen, als auch für die Evaluation der Methoden und Werkzeuge verwendet.

Themenbereiche, die in dieser Arbeit nicht mehr behandelt werden können, aber wertvolle Beiträge für eine Weiterentwicklung bieten, sind insbesondere die Usibility-Betrachtung der generierten Benutzerschnittstellen, Performance-Analysen, und das User-Interface-Design. In dieser Arbeit wird ein Verfahren für die automatisierte Generierung von Benutzerschnittstellen entwickelt und evaluiert. Die Bereitstellung einer vollständigen Werkzeugunterstützung (Modellierung, Generierung, Nachbearbeitung der generierten Benutzerschnittstelle) kann im Rahmen dieser Arbeit nicht erfolgen, wohl aber eine Spezifikation der benötigten Werkzeuge.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich thematisch in drei Abschnitte: Vorarbeiten, Modellierung und Generierung. In dem folgenden **Kapitel 2** werden aktuelle Arbeiten und Projekte im Umfeld der Generierung von Benutzerschnittstellen vorgestellt. Dabei wird untersucht, welche Strategien verfolgt werden, um diese Zielsetzung zu erreichen. Intensiver werden Möglichkeiten zur Spezifikation von Benutzerschnittstellen durch XML-basierte Beschreibungssprachen betrachtet, die Teil der in der Arbeit entwickelten Methodik sein werden.

Der entwickelte Generierungsprozess wird anhand des Anwendungsfeldes „patientenorientierte telemedizinische Dienste“ evaluiert. In **Kapitel 3** wird dieses Anwendungsfeld motiviert und dargestellt, warum es für die Problemstellung der Arbeit besonders geeignet ist. Weiterhin wird der Adipositas-Begleiter als Beispiel einer solchen telemedizinischen Anwendung vorgestellt, anhand dessen die entwickelten Modelle, Werkzeuge und Methoden dieser Arbeit evaluiert werden. Dieses Anwendungsbeispiel wird sich als „roter Faden“ durch die gesamten folgenden Kapitel ziehen.

Da die Generierung von Benutzerschnittstellen sehr unterschiedliche Themenbereiche wie die Interaktionsmodellierung, Kontextmodellierung und schließlich die Spezifikation und auch Ausführung von Be-

nutzerschnittstellen umfassen wird, erfolgt in **Kapitel 4** vorgreifend eine ganzheitliche Betrachtung des Generierungsprozesses.

In **Kapitel 5, 6 und 7** werden jeweils die drei Kernmodelle „Interaktionsmodell“, „Kontextmodell“ und die „Spezifikation der Benutzerschnittstellenbausteine“ erarbeitet. Dabei ist jedes Kapitel in die folgenden drei Sektionen unterteilt:

- Anforderungsanalysen
  In einem ersten Schritt erfolgt eine Analyse des Problemfeldes und die Betrachtung existierender Arbeiten in diesem Bereich.
- Spezifikationen
  Dieser Abschnitt definiert den generischen Anteil der Arbeit, in welchem Modelle und Spezifikationen erarbeitet werden, die auf den vorher definierten Anforderungen basieren.
- Konkretisierung am Beispiel der „patientenorientierten telemedizinischen Dienste“
  In diesem Abschnitt erfolgt die Konkretisierung der vorangegangenen Arbeiten und eine Evaluation anhand des eingeführten Anwendungsbeispiels.

Basierend auf den entwickelten Modellen werden in **Kapitel 8** Verfahren, Methoden und Werkzeuge für eine automatische Generierung von Benutzerschnittstellen erarbeitet und anhand des Anwendungsbeispiels evaluiert.

In **Kapitel 9** erfolgt die Evaluation in Form eines „Proof of Concept“, indem das entwickelte Verfahren für die Generierung unterschiedlicher Benutzerschnittstellen eingesetzt wird. Im Rahmen dieses Kapitels werden die Ergebnisse des Generierungsprozesses präsentiert und die Leistungen des Verfahrens bewertet. Anschließend folgt in **Kapitel 10** eine Zusammenfassung und Diskussion der Arbeit.

2 Modellbasierte Ansätze für die Benutzerschnittstellengenerierung

Die Konzeption und Entwicklung von Werkzeugen für die Erstellung von graphischen Benutzerschnittstellen ist eine der Kernaufgaben der modernen Softwareentwicklung. Nahezu alle heute eingesetzten Benutzerschnittstellen wurden mit Hilfe von Werkzeugen wie GUI-Toolkits (MFC oder WFP von Microsoft, Swing für Java, VCL für Borland Delphi) und den in die IDE (Integrated Development Environment) integrierten GUI-Editoren erstellt.

Auch modellbasierte Ansätze für die Benutzerschnittstellengenerierung verstehen sich als solche Werkzeuge, die aber eine automatisierte Generierung von Benutzerschnittstellen ermöglichen. Wie der Name bereits impliziert, verlangen solche Systeme die Definition unterschiedlicher Modelle, aus denen schließlich durch Ableitungen und Modelltransformationen eine ausführbare Benutzerschnittstelle generiert wird. Zielsetzung dieses Kapitels ist es, einen Einblick in den aktuellen Stand existierender Systeme für die modellbasierte Generierung von Benutzerschnittstellen zu geben und deren Konzepte und Methoden zu erläutern. Eine vertiefende Diskussion über einzelne Modelle und wie diese spezifiziert werden, kann im Rahmen dieser Betrachtung nicht erfolgen. Im Anhang E dieser Arbeit finden sich über 40 Projekte, in denen solche Modelle und Spezifikationen vorgestellt werden, wobei auch diese Liste keinen Anspruch auf Vollständigkeit erheben kann. Eine ausführliche Betrachtung der für diese Arbeit relevanten Modelle und deren Spezifikationsmöglichkeiten wird in dem jeweiligen Kapitel durchgeführt, in welchem die einzelnen Modelle für den „Compositional Modeling“-Ansatz erarbeitet werden. Dies sind Interaktionsmodelle in Abschnitt 5.1, Kontextmodelle in Abschnitt 6.1 und Modelle für die abstrakte Beschreibung von Benutzerschnittstellen in Abschnitt 7.2.

Modellbasierte Ansätze können auf eine lange Geschichte verweisen. Bereits in den 80er Jahren wurden die ersten Systeme für die modellbasierte Generierung von Benutzerschnittstellen vorgestellt (vgl. [MHF00]). Viele der damaligen Arbeiten haben Grundlagen geschaffen, die heutige Forschungsarbeiten noch stark beeinflussen. In einem ersten Schritt wird daher ein kurzer historischer Rückblick gewährt, um aufzuzeigen, welche Ansätze weiterentwickelt wurden, und welche sich nicht bewähren konnten.

Aus dieser historischen Betrachtung heraus wird aufgezeigt, wie sich das aktuelle Verständnis modellbasierter Systeme für die Benutzerschnittstellengenerierung geändert hat und wie moderne Lösungen aufgebaut sind. Hierzu wird das sogenannte „Unified Reference Framework“ vorgestellt, das als Vorlage und als gemeinsame Beschreibungsebene für unterschiedliche Systeme fungiert. Die Vorstellung der aktuellen Arbeiten ist in zwei Schritte unterteilt. Im ersten Schritt werden sogenannte „Multi-Layer“-Systeme beschrieben, die sich unterschiedlicher Abstraktionsebenen bedienen und in der Forschung als „State of the Art“ akzeptiert sind. In einem zweiten Schritt wird anhand von kommerziellen Lösungen präsentiert, welche Bereiche der aktuellen Forschungsarbeiten bereits einen Weg in die Praxis gefunden haben und wie sie dort umgesetzt wurden.

Abschließend wird anhand der vorgestellten Systeme diskutiert, wie sich ein „Compositional Modeling“-Ansatz in die aktuellen Entwicklungen (sowohl aus Sicht der Forschung, als auch der Praxis) integrieren lässt.

2.1 Eine historische Betrachtung

Die Forschung im Bereich der Entwicklung von graphischen Benutzerschnittstellen hat für die moderne Softwareentwicklung einen hohen Stellenwert. Myers zeigt in seiner historischen Betrachtung von Forschungsaktivitäten im Bereich der Benutzerschnittstellenentwicklung (vgl. [MHF00] und [Mye98]), wie wissenschaftliche Arbeiten der 60er und 70er Jahre Grundlagen (Maus, Window-Systeme und -Toolkits, Event-basierte Systeme, Hypertext (erstes Konzept 1945 in [Bus45])) für noch heute aktuelle Systeme geschaffen haben. Er führt aber auch an, dass Versuche der formalen Beschreibung von Benutzerschnittstellen, wie sie in den 80er Jahren diskutiert wurden, heute kaum mehr praktische Relevanz besitzen. Beispiele für Systeme, die formale Spezifikationen verwenden, sind:

User Interface Management Systeme (kurz UIMS) definieren Ansätze für eine formale Spezifikation von Benutzerschnittstellen, aus der ausführbare Anwendungen generiert werden können. Der Begriff UIMS ist dabei in Analogie zu DBMS (Database Management System) entstanden und soll die Idee eines DBMS für die Verwaltung und Organisation von Datenbeständen auf die Benutzerschnittstellengenerierung übertragen. Eine eingehende Betrachtung von UIMS findet sich in Abschnitt 5.1.4.1.

Dialog-basierte Systeme nutzen Dialogspezifikationen (vgl. Abschnitt 5.2.4.2), um Benutzerschnittstellen auf einem höheren Abstraktionslevel (als bei der Programmierung) zu beschreiben. Zielsetzung ist es, dass für die Entwicklung von Benutzerschnittstellen keine Programmierkenntnisse mehr notwendig sind, sondern lediglich Kenntnisse über die verwendete Dialogspezifikation. Dialogspezifikationen nutzen unterschiedliche Modellierungsmethodiken wie Zustandsübergangsdiagramme, Petri-Netze, Graphtransformationen, Grammatiken, aber auch Event-basierte Beschreibungssprachen und Constraints werden für die Modellierung eingesetzt. Eine eingehende Betrachtung von Dialog-basierten Systemen und deren Modellierungsmethodiken findet sich in Abschnitt 5.2.4.2.

Modellbasierte Ansätze (vgl. [Pue97], [Sze96]) für die Generierung von Benutzerschnittstellen erweitern den dialogzentrierten Ansatz, der auch den UIMS zugrunde liegt, um weitere Aspekte der Benutzerschnittstellenbeschreibung durch Domänen-, Aufgaben-, Nutzer-, Dialog- und Präsentationsmodelle (vgl. [PE02]). Während sich UIMS lediglich mit der Spezifikation der Interaktionen begnügen, und aus der Semantik der Modellierungsmethodik eine entsprechende Benutzerschnittstelle generieren, setzen modellbasierte Systeme eine Vielzahl von Modellen ein, die unterschiedliche Aspekte einer Benutzerschnittstelle beschreiben. Die modellierten Aspekte werden zueinander in Beziehung gesetzt, und bilden eine Informationsbasis, aus welcher eine Generierung der Benutzerschnittstelle möglich wird. Welche Modelle für die Generierung von User Interfaces eingesetzt werden, hängt von den verwendeten Systemen (vgl.[MFC01], [FDR*04], [GMP*98]) und den zu modellierenden Anwendungen ab. Den meisten modellbasierten Systemen gemeinsam sind das Dialog-, das Präsentations- und das Domänenmodell. Mo-

dellbasierte Systeme zeichnen sich durch deklarative Modelle interaktiver Anwendungen aus (vgl. [Sch94]), mittels derer ausführbare Systeme generiert werden können.

Myers (vgl. [MHF00] und [Mye98]) führt das Scheitern dieser Systeme auf zwei Entwicklungen zurück:

- Die zunehmende Vereinheitlichung von graphischen Benutzerschnittstellen durch das WIMP (Windows Icon Menu, Pointer)-Paradigma führte zu einer Stagnation in der Entwicklung von innovativen Benutzerschnittstellen (vgl. [GM91]). Durch diese Standardisierung wurden aber die Vorteile einer höheren Abstraktionsebene, auf der Benutzerschnittstellen beschrieben werden, weitgehend unnötig gemacht. Entsprechend boten Toolkits; die auf unterschiedlichen Plattformen verfügbar waren, und auch Web-Browser, welche in der Lage waren, Hypertext zu interpretieren, ausreichend gute Lösungen. Der Bedarf für eine flexible Generierung war nicht mehr vorhanden (moving target problem).
- Die Generierung von Benutzerschnittstellen hatte zum Ziel, die Rolle des Interface Designers weitgehend zu ersetzen, da die Konzeption und auch die Realisierung aus der gegebenen, formalen Spezifikation erfolgen sollte. Es zeigte sich aber, dass die Qualität der generierten Benutzerschnittstellen nicht mit den manuell erstellten Benutzerschnittstellen konkurrieren konnte. Insbesondere der Nachteil, dass die manuelle Nachbearbeitung der generierten Benutzerschnittstellen wieder den Einsatz von Programmiersprachen erforderte, relativierte den Vorteil der automatischen Generierung. Ein weiterer Nachteil ist, dass manuell durchgeführte Änderungen verloren waren, wenn die Benutzerschnittstelle erneut (oder mit veränderten Eingabeparametern) generiert wurde.

Myers betonte weiter, dass auch die frühen modellbasierten Ansätze (wie beispielsweise Jade (vgl. [ZM90]), UIDE (vgl. [SFG93]), ITS (vgl. [WBB*90])) an derselben Problemstellung gescheitert sind, zukünftig aber wieder an Bedeutung gewinnen werden. Er begründet diese Prognose durch die zunehmende Diversität von Benutzerschnittstellen, die sich aus den folgenden Entwicklungen ergeben: eine stetig wachsende Zahl mobiler Endgeräte und eingebetteter Systeme; durch neue Möglichkeiten zur Interaktion mit Computersystemen (Gestenerkennung, Handschrifterkennung und „multimodale Interaktionen“[1]); dem Anspruch, dass zukünftige Systeme mehr auf den Menschen ausgerichtet werden müssen und sich daher stärker unterscheiden werden.

Die Vorhersagen, die Meyer im Jahr 2000 getroffen hat, haben sich weitgehend bewahrheitet. Acht Jahre später haben modellbasierte Systeme wieder an Bedeutung gewonnen und gelten als akzeptierter und vielversprechender Ansatz für Innovationen im Bereich der Benutzerschnittstellengenerierung (vgl. [SJGL08]).

[1] Mulitmodale Interaktionen beschreiben Eingaben, die sich gleichzeitig mehrerer Modalitäten wie Tastatur, Maus, aber auch Sprache, Gesten und Touch-Screens bedienen.

Neben der zunehmenden Diversität der Benutzerschnittstellen lassen sich zwei weitere Trends identifizieren, welche modellbasierte Systeme für die Benutzerschnittstellengenerierung weiter in den Fokus aktueller Forschungsarbeiten rücken.

Die Etablierung von XML als einheitliche und allgemein akzeptierte Beschreibungssprache führte zu einem Schub in der Entwicklung neuer Spezifikationen von Benutzerschnittstellenmodellen (vgl. [SV03]). Einhergehend mit der Entwicklung dieser Spezifikationen folgen auch Werkzeuge, um diese für die Benutzerschnittstellengenerierung nutzbar zu machen.

Forschungsgruppen und Initiativen im Bereich der MDA (Model Driven Architecture), versprechen eine Verbesserung (Portierbarkeit, Wiederverwendung, Reduktion der Kosten) im Bereich der Softwareentwicklung und sind dabei, sich in der kommerziellen Softwareentwicklung zunehmend zu etablieren (vgl. [GPR06]). Auch die Betrachtung von Benutzerschnittstellen muss dabei mit den Konzepten der MDA einhergehen. Modellbasierte Systeme sind offensichtlich besonders für MDA Ansätze geeignet. Wie modellbasierte Systeme für die Benutzerschnittstellengenerierung im Rahmen der MDA eingesetzt werden können, wird beispielsweise in [CVC08] und [MS09] präsentiert, um einen ganzheitlichen modellbasierten Ansatz zu verfolgen.

In den letzten 5 Jahren sind eine immense Anzahl von Spezifikationssprachen für Benutzerschnittstellenmodelle entwickelt worden. Im Anhang E dieser Arbeit findet sich eine Liste von über 40 solcher Arbeiten und Projekte sowohl aus der Forschung, als auch aus der Wirtschaft. Entnommen ist diese Liste einer Web-Page des Projektes UsiXML (vgl. [LVM*05]), das sich selbst mit der Erstellung von XML-basierten Spezifikationen von Benutzerschnittstellen beschäftigt und versucht hat, einen ersten Überblick über ähnliche Systeme zu erstellen. Diese Liste umfasst dabei nur XML-basierte Spezifikationen und erhebt keinen Anspruch auf Vollständigkeit.

2.2 Aufbau moderner modellbasierter Systeme

Bevor die ersten modellbasierten Systeme für die Generierung von Benutzerschnittstellen vorgestellt werden, wird in einem ersten Schritt versucht, ein allgemeines Verständnis für den Aufbau solcher Systeme zu schaffen. Die Grundlage dieser Betrachtung bildet das in [GCT*03] vorgestellte „Unified Reference Framework“ (kurz Reference Framework), das als einheitliche Beschreibung modellbasierter Ansätze in der Wissenschaft weitgehend akzeptiert ist (vgl. [PSMM*08], [HSJ09], [VP08]). Das Reference Framework beschreibt grundlegende Konzepte und Methodiken existierender modellbasierter Ansätze und definiert einen einheitlichen Rahmen für ihre Beschreibung.

Abbildung 2 visualisiert das „Reference Framework“, das sich in drei Bereiche unterteilen lässt. Einerseits das „Ontological Model“ (links in Abbildung 2), das „Archetypal Model“ (zwei Bereiche rechts in Abbildung 2) und schließlich der „Execution Process“ (unterer Bereich in Abbildung 2).

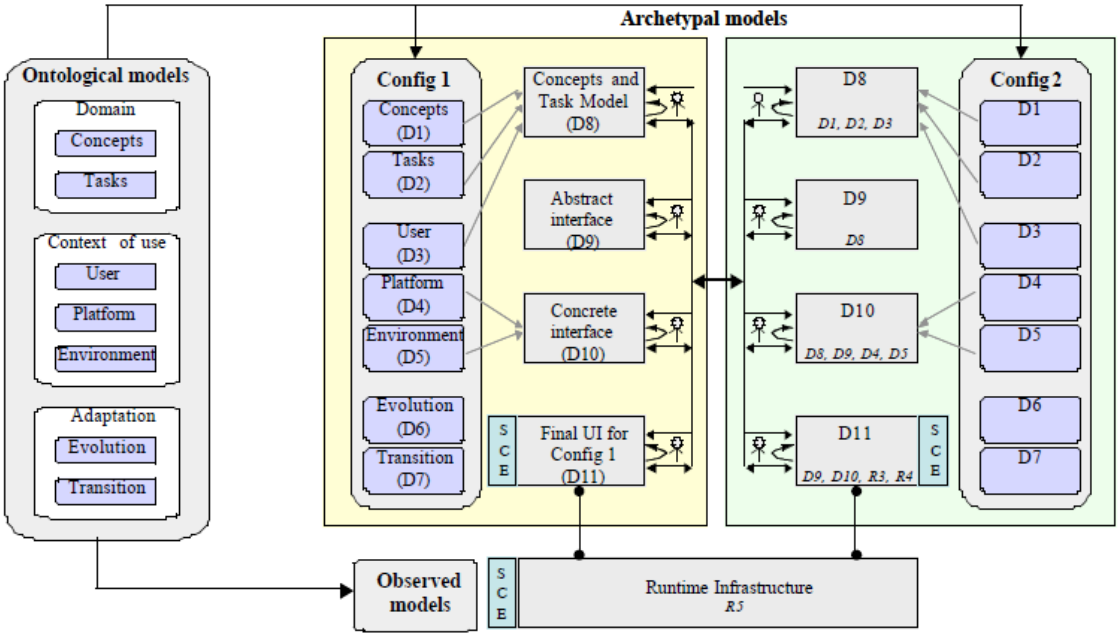


Abbildung 2: Unified Reference Framework (vgl. [GCT*03])

Die „Ontological Models“ werden in [GCT*03] als Meta-Modelle beschrieben, welche unabhängig von jeder Domäne sind, aber auf eine gegebene Fragestellung (im Falle des Reference Frameworks die Benutzerschnittstellengenerierung) hin ausgerichtet sind. Damit folgen die „Ontological Models“ der Definition, die in [Grub95] gegeben ist. Aus dieser Perspektive werden drei relevante Modelle entwickelt: ein Domain Model für die Beschreibung von Konzepten und Aufgaben eines Nutzers in einer Domäne; ein Context Model, das den Benutzer, die Plattform und die Umgebung der Nutzung beschreibt; ein Adaption Model, das Auswirkungen im Falle eines Kontextwechsels beschreibt.

Die „Archetypal Models“ („Config 1“ und „Config 2“) verstehen sich als Instanzierung der „Ontological Models“, welche auf Basis der beschriebenen Ontologien erstellt werden. „Config 1“ und „Config 2“ in Abbildung 2 visualisieren dabei zwei unterschiedliche Instanzierungen. Die „Archetypal Models“ werden im Rahmen des Frameworks auch initiale Modelle genannt, da sie aus Sicht eines Generierungsprozesses einer Benutzerschnittstelle den ersten Modellierungsschritt beschreiben. Jeweils rechts neben den beiden „Archetypal Models“ sind die sogenannten transienten Modelle (Concepts and Task-Model, Abstract User Interface, ...), die im Laufe des Entwicklungsprozesses erzeugt werden. Transiente Modelle sind Zwischenprodukte, die mit dem Ziel erstellt wurden, eine ausführbare Benutzerschnittstelle (Final UI) zu erzeugen. Bei den Transformationen zwischen den transienten Modellen und den „Archetypal Models“ wird zwischen vertikalen und horizontalen Transformationen unterschieden:

- Die vertikale Transformation beschreibt einen Konkretisierungsprozess, bei dem ausgehend von einer abstrakten Beschreibung durch einzelne Konkretisierungsschritte schließlich eine ausführbare Beschreibung erreicht wird. Weiterhin können sie auch einen Abstraktionsprozess beschreiben, der vor allem im Zusammenhang mit dem Reverse-Engineering eingesetzt wird. Dieser erlaubt es, aus kon-

kreten Beschreibungen Abstraktionen zu gewinnen, die gegebenenfalls auch Auswirkungen auf die initialen Modelle haben können.

- Die horizontale Transformation beschreibt die Nutzung von initialen Modellen für die Generierung von transienten Modellen. Dabei werden Transformationen zwischen Modellen auf der gleichen Abstraktionsebene beschrieben.

Das „Reference Framework" trifft keine Aussagen darüber, welche initialen Modelle eingesetzt werden. Abbildung 2 veranschaulicht einen möglichen Modellsatz. Wie in [GCT*03] beschrieben wird, unterscheiden sich modellbasierte Systeme für die Generierung von Benutzerschnittstellen insbesondere durch die Wahl der verwendeten Modelle.

Das Reference Framework definiert auch keinen Generierungsprozess. So muss die Modellierung nicht bei einem „Konzept- und Aufgabenmodell" beginnen. In [GCT*03] finden sich auch Beispiele (vgl. [PS03]), die direkt mit der Modellierung der abstrakten Benutzerschnittstelle starten und gänzlich auf eine konzeptionelle Beschreibung verzichten.

Auch wenn kein Prozess vorgegeben ist, spezifiziert das Reference Framework mögliche Abstraktionsebenen, auf denen eine Beschreibung erfolgen kann. Die folgenden Abstraktionsebenen sieht das Reference Framework vor:

- Das „Concept and Task-Model" (kurz C&T) beschreibt die Aufgaben und Unteraufgaben, die ein Benutzer ausführen muss, und domänenspezifische Konzepte, soweit sie für die Durchführung der definierten Aufgaben benötigt werden. Ein Beispiel für ein Task-Modell und die Semantik der Modellierung wird in Abschnitt 5.1.4.2 vorgestellt.
- Das „Abstract User Interface" (kurz AUI) wird als eine kanonische Wiedergabe von Aufgaben und domänenspezifischen Konzepten beschrieben, die unabhängig von Interaktionsobjekten[2] und Modalitäten ist. Auf Ebene der AUI erfolgt eine Gruppierung von abstrakten Interaktionsobjekten in Containern (z.B. Fenster, Masken), auf deren Basis Strukturen wie Gruppierungen, grundlegende Abläufe und Beziehungen zwischen den Containern und den abstrakten Interaktionsobjekten definiert werden können. Dabei ist eine Beschreibung auf dieser Ebene unabhängig von Modalitäten und sollte allgemeingültig sowohl für eine graphische Benutzerschnittstelle als auch für sprachgesteuerte oder 3D-Umgebungen gelten.
- Das „Concrete User Interface" (kurz CUI) beschreibt eine Benutzerschnittstelle unabhängig von einer Programmiersprache. CUIs sind aber bereits auf eine konkrete Plattform (fensterbasierte Desktop-Anwendung, mobile PDA-Anwendung, usw.) ausgerichtet. Die CUI benötigt noch eine Art

[2] Interaktionsobjekte beschreiben Elemente der Benutzerschnittstelle, die ein Benutzer manipulieren kann, während Modalitäten die Art und Weise der Manipulation beschreiben. Interaktionsobjekte können beispielsweise Button, Listen oder Menüs in fensterbasierten Systemen sein, aber auch Sprachkommandos in sprachbasierten Benutzerschnittstellen.

Framework, das zusätzliche Informationen und Methodiken liefert, um eine ausführbare Benutzerschnittstelle zu generieren.

- Das „Final User Interface“ (kurz FUI) beschreibt die ausführbare Benutzerschnittstelle, die auf einer realen Plattform entweder durch Interpretation (z.B. durch einen Web-Browser) oder direkt (z.B. nach Kompilierung) ausgeführt werden kann.

Die beschriebenen Abstraktionsebenen haben nicht nur Einfluss auf die Beschreibungsdetaillierung, sondern auch auf die Art der verwendeten Spezifikation. Während im C&T neben der hierarchischen Dekomposition einfache Ablaufmodellierungen verwendet werden, die das Zusammenspiel von Aufgaben beschreiben, nutzen AUI schon komplexere Spezifikationsmethodiken, wie beispielsweise Petri-Netze oder Zustandsübergangsgraphen. Auf Ebene der CUI wiederum werden vermehrt Event-basierte Dialogspezifikationen verwendet, die sehr mächtig in ihrer Beschreibung sind, sich aber nur schwer analysieren lassen. Auf Ebene des FUI werden schließlich tatsächlich plattformabhängige Methoden (Programmcode) verwendet.

Der letzte Bereich des Reference Frameworks (der „Execution Process“) umfasst nicht nur die Ausführung der FUI, sondern definiert durch die „Observed Models“ weiterhin die Möglichkeit, auch in die Ausführung der Benutzerschnittstelle einzugreifen, wenn Änderungen im Kontext der Nutzung festgestellt wurden. Das Reference Framework definiert hier einen Regelkreis, der in die Ausführung eingreift, sobald ein relevanter Kontextwechsel stattgefunden hat.

Das Reference Framework definiert wesentliche Bausteine, mit denen modellbasierte Systeme für die Benutzerschnittstellengenerierung beschrieben werden können. Diese Bausteine sind dabei als Platzhalter für Konzepte, Spezifikationen und Werkzeuge zu verstehen, in denen sich die unterschiedlichen Systeme schließlich unterscheiden.

2.3 Multi Layer Systeme

Das im vorgehenden Abschnitt eingeführte Reference Framework definiert vier Abstraktionsebenen (Task, AUI, CUI und FUI) für die Beschreibung von Benutzerschnittstellen, die im Generierungsprozess eingesetzt werden. Im Gegensatz zu Modellen und Spezifikationsmethoden, die von System zu System sehr unterschiedlich gestaltet sein können und nicht direkt vergleichbar sind, liefern diese Abstraktionsebenen eine Möglichkeit, modellbasierte Ansätze zu klassifizieren.

In den folgenden Abschnitten werden Systeme vorgestellt, die Spezifikationen für alle vier Abstraktionsebenen anbieten, und damit auch zu den komplexeren modellbasierten Systemen gezählt werden können.

Es existiert eine Vielzahl von Projekten, in denen solche Systeme (vgl. bspw. XIML [FDR*04], ISML [CH03], SEESCOA [BLC03], UsiXML [LVM*04], Teresa [MPS04]) entwickelt wurden. Auch sind vergleichende Gegenüberstellungen (vgl. bspw. [SV03], [Dra05], [GCVA09]) der Systeme in der Literatur zu finden, welche Bewertungen nach unterschiedlichen Kriterien durchführen. Im Rahmen dieser Arbeit wurden drei Systeme ausgewählt, die stellvertretend für die übrigen vorgestellt werden.

Als erstes System wird Teresa (vgl. [MPS04]) vorgestellt, welches vereinfacht als modellbasiertes Autorensystem bezeichnet werden kann. Teresa nutzt Modelle und abstrakte Spezifikationen von Benutzerschnittstellen primär als Werkzeug, mit dem ein Modellierer Benutzerschnittstellen beschreiben kann.

UsiXML (vgl [LVM*04]), welches als zweites präsentiert wird, bietet ebenfalls die Möglichkeit, dass Modellierer in den Generierungsprozess eingreifen können. UsiXML fokussiert aber eine automatisierte Generierung aus den Modellen heraus, die durch Transformationsregeln zwischen Modellen und den unterschiedlichen Abstraktionsebenen einer Benutzerschnittstelle definiert werden.

Während Teresa und UsiXML für die Spezifikation von Modellen und auch der verwendeten Abstraktionsebenen XML verwenden, wird als dritter modellbasierter Ansatz ein System aus der Domäne des Model Driven Web-Engineerings (vgl. [WD08]) vorgestellt.

UWE (UML-based Web Engineering, vgl. [Koc07]) zeichnet sich dadurch aus, dass nicht nur eine Benutzerschnittstelle spezifiziert wird, sondern eine vollständige Web-Anwendung generiert wird. Auch in diesem Anwendungsbereich existieren unterschiedliche Ansätze (vgl. bspw. WebML [BCFM07], OOHDM [RS06]). UWE wurde für diese Arbeit ausgewählt, weil es einen (im Vergleich zu anderen Systemen) ausgeprägten Fokus auf die Beschreibung der Benutzerschnittstelle legt.

2.3.1 Teresa/MARIA

Teresa (Transformation Environment for Interactive Systems Representation) versteht sich als modellbasierte Umgebung zur Generierung von Benutzerschnittstellen für verschiedene Endgeräte (vgl. [MPS04], [BCP04]). Für die Generierung wird ein Top-Down Ansatz verwendet, in welchem Benutzerschnittstellen in einem ersten Schritt auf einer hohen Abstraktionsebene modelliert werden. Anschließend wird schrittweise die abstrakte Beschreibung der Benutzerschnittstelle durch Modelltransformationen konkretisiert. In Teresa sind die folgenden Abstraktionsebenen vorgesehen:

- „High-level Task-Model“
  Es wird ein einziges Modell spezifiziert, das die folgenden Bereiche umfasst: Ziele, Aufgaben und auch Rollen der beteiligten Nutzer und schließlich die Domäne, bestehend aus Objekten (und deren Beziehungen), die notwendig für die Durchführung des Anwendungsfalls sind. Für die Modellierung wird das Task-Modell (Concurrend Task Tree, vgl. [Pat99]) eingesetzt, wie es in Abschnitt 5.1.4.2 eingehend beschrieben wird.
- „System Task-Model“
  Dieses Modell beschreibt eine Konkretisierung des vorhergegangenen Modells auf ein bestimmtes Endgerät hin. Durchgeführt werden Anpassungen des „High-level Task-Model“, die notwendig sind, um die dort beschriebenen Aktivitäten und Objekte zu realisieren, indem Tasks ausgetauscht oder aufgespalten werden (wenn sie sich beispielsweise nicht als ein Schritt auf der Zielplattform realisieren lassen).

- „Abstract User Interfaces“
  Ermittelt wird eine abstrakte Beschreibung der Benutzerschnittstelle aus dem „System Task-Model“, indem die einzelnen Tasks auf sogenannten „Presentation Task Sets“ abgebildet werden, die beschreiben, durch welche Präsentationen Tasks realisiert werden können. Die einzelnen Präsentationen können weiterhin durch zusätzliche Relationen (Grouping, Relation, Ordering, Hierarchy) zu einer Präsentation verknüpft werden.
- „Executable User Interface“
  Die ausführbare Benutzerschnittstelle wird generiert, indem die ermittelten Präsentationen auf konkrete Benutzerschnittstellenelementen (Widgets), die auf der jeweiligen Plattform verfügbar sind, abgebildet werden.

Abbildung 3 veranschaulicht die dargestellten Modelle und zeigt, auf welche Weise unterschiedliche Benutzerschnittstellen generiert werden können.

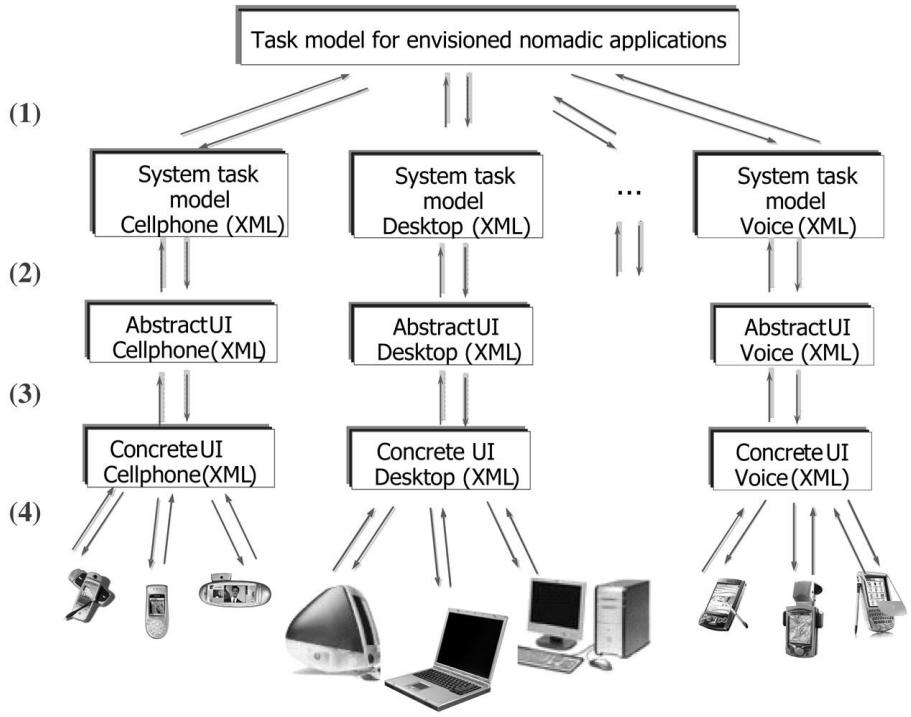


Abbildung 3: Teresa Generierungsprozess aus [MPS04]

Teresa zeichnet sich durch eine starke Werkzeugunterstützung aus. So werden für jeden Schritt interaktive graphische Werkzeuge angeboten, welche bei der Erstellung der Modelle helfen. Für jedes der Modelle existieren XML-basierte Spezifikationen, so dass Zwischenergebnisse exportiert oder auch manuell bearbeitet werden können. Der Entwickler ist dabei in den gesamten Prozess (vom Task-Modell bis zur ausführbaren Benutzerschnittstelle) eingebunden und verfeinert die Modelle mit den zur Verfügung gestellten Werkzeugen.

Aktuell wird Teresa in den Systemen MARIA (vgl. [PSS09]) und „Multimodal Teresa“ (vgl. [PSMM*08]) weiterentwickelt. MARIA versteht sich dabei als konsequente Weiterentwicklung und Spezialisierung mit dem Fokus, Benutzerschnittstellen für Web-Services erstellen zu können. Folgende Erweiterungen wurden dabei eingeführt:

- Ein Datenmodell wurde eingeführt, das Daten beschreibt, die durch die Benutzerschnittstelle erstellt oder manipuliert werden. Das Datenmodell ermöglicht es, Bedingungen und Beziehungen zwischen Präsentationen zu formulieren.
- Ein Event-Modell wurde eingeführt, das es ermöglicht, Präsentationen über Events zu modifizieren oder auch externe Systeme anzusprechen.
- Es wurde die Möglichkeit geschaffen, „Bereiche der Präsentation“ kontinuierlich mit externen Systemen zu verbinden, um beispielsweise Änderungen in einer Datenbasis direkt auf der Benutzerschnittstelle darstellen zu können.
- Es wurde die Möglichkeit geschaffen, „dynamische Bereiche“ in der Benutzerschnittstelle zu definieren, welche es erlauben, dass nur Teile einer Benutzerschnittstelle „nachgeladen“ werden.
- Eine Web-Service Schnittstelle für Datenflüsse wurde eingeführt.

Ziel von „Multimodal Teresa“ ist es, die existierenden, nur auf graphische Benutzerschnittstellen ausgerichteten Präsentationen zu erweitern. Vorgesehen ist in einem ersten Schritt die Einführung von interaktiven Vektorgraphiken, die parallele Nutzung von Sprachsteuerung und graphischen Benutzerschnittstellen (Multimodalität), Interaktionen über Set-Top-Boxen und Gestensteuerung. „Multimodal Teresa“ führt die hierfür benötigten Konstrukte ein und erweitert die existierenden Werkzeuge, die Teresa bisher bereitstellt.

2.3.2 UsiXML

UsiXML (User Interface Extensible Markup Language) realisiert einen modellbasierten Ansatz für die Beschreibung von Benutzerschnittstellen auf unterschiedlichen Abstraktionsebenen (vgl. [LVM*04]). Die Spezifikation von Benutzerschnittstellen kann dabei aus unterschiedlichen Richtungen (Multi-Path-Development) erfolgen. Im Gegensatz zum Teresa Ansatz, der auf einer hohen Abstraktionsebene beginnt und dann schrittweise konkretisiert wird, ermöglicht UsiXML die Spezifikation auf jeder Abstraktionsebene oder bei beschreibenden Modellen (User, Domäne oder Plattform). So kann die Beschreibung auf Ebene eines Dialogmodells beginnen, aus dem dann Tasks abgeleitet werden können, aus denen schließlich Objekte (Datenobjekte) ermittelt werden, die das Dialogmodell beeinflussen. UsiXML sieht die Definition von Regeln vor, welche die unterschiedlichen Modelle miteinander verknüpfen und sicherstellen, dass diese zueinander konsistent bleiben. Dabei bietet UsiXML eine XML-basierte Spezifikation für unterschiedliche Modelle (vgl. [LVM*04]):

„Task-Model“, das Ziele, Aufgaben und Unteraufgaben für die Durchführung einer Interaktion beschreibt (vgl. Abschnitt 5.1.4.2). Wie bei Teresa werden CTTs (Concurrent Task Trees) für die Modellierung verwendet, die um zusätzliche Konstrukte erweitert wurden.

„Domain Model“, das Objekte (Daten und Attribute) einer Anwendungsdomäne beschreibt (beispielsweise Flugplan). Für die Beschreibung werden UML Klassendiagramme verwendet. Zusammen mit dem Task-Model lassen sich dadurch beispielsweise das Erzeugen, die Manipulation, das Lesen oder das Updaten von Objekten beschreiben.

„Context Model“, das Beschreibungen des Nutzers, der Hard- und Software-Plattform und der Umgebung der Nutzung enthält. Die Beschreibung erfolgt in Anlehnung an W3C CC/PP Profile (vgl. [Kiss07]), wobei lediglich das CC/PP Vokabular eingesetzt wird, für die Spezifikation aber eine eigene XML-basierte Beschreibung genutzt wird.

„Abstract User Interface“, das ein Modell einer Benutzerschnittstelle spezifiziert. Interaktionen werden basierend auf der Darstellung und der Manipulation von Objekten des Domain Models geräte- und kontextunabhängig beschrieben. Dieses Modell setzt sich dabei aus „Abstract Interaction Objects“ (kurz AIO) und „Abstract User Interface Relationships“ (AUI Relationship) zusammen. AIOs beschreiben dabei Interaktionsobjekte (z.B. Window, Button, Sprachein- und Sprachausgabe) auf einer höheren Abstraktionsebene (Container, Control, Selection, Output, Input ...). AUI Relationships beschreiben schließlich Beziehungen zwischen den AIOs. Beziehungen können dabei Dialogtransitionen (Suspend, Resume, Enable, Disable), Gruppierungen (zusammenstellen von Objekten zu einem „Interaction Space“) oder aber zeitliche Abhängigkeiten (gleichzeitig starten, gleichzeitig beenden, folgend, usw.) sein.

„Concrete User Interface“, das ein Modell einer Benutzerschnittstelle beschreibt. Für die Beschreibung werden Abstraktionen von Interaktionsobjekten (Button, List, Menu, Text, Window, Dialog...) benutzt. Zusätzlich ist ein Dialogmodell spezifiziert, welches das dynamische Verhalten der Benutzerschnittstelle beschreibt. Das Dialogmodell verwendet dabei eine Event-basierte Spezifikation und bietet zusätzlich eine eigene Sprache für die Beschreibung von Navigationsstrukturen an.

„Mapping Model“, das die Definition von Beziehungen zwischen den unterschiedlichen Modellen ermöglicht. Dabei werden vordefinierte Beziehungstypen verwendet, bspw. Task auf Domain, Domain auf Präsentation, Task auf Abstract User Interface, usw.

„Transformation Model“, das die einzelnen Regeln für die Transformation eines Modells in ein anderes enthält. Diese Regeln werden als Graph Grammatiken (vgl. [Roz97]) formuliert und realisieren das Werkzeug, mit dem ausführbare Benutzerschnittstellen generiert werden können.

Diese Betrachtung der einzelnen Modelle verdeutlicht die Komplexität modellbasierter Systeme, wenn sie für eine automatisierte Generierung von Benutzerschnittstellen eingesetzt werden. UsiXML versucht dieser Problemstellung zu begegnen, indem Modellierungswerkzeuge zur Erstellung der einzelnen Modelle und auch der Transformationsregeln angeboten werden (bspw. IdealXML (vgl. [ML06]) für das

Task-Model oder GraphiXML (vgl. [MV08]) für das „Concrete User Interface“). Für die Beschreibung einer Benutzerschnittstelle in UsiXML müssen nicht alle diese Modelle verwendet werden, es ist aber eine Erweiterung um zusätzliche Modelle vorgesehen.

Die Modelle, die in UsiXML verwendet werden, nutzen unterschiedliche Methoden für ihre Beschreibung: beispielsweise Task-Trees für das Task-Model, UML Klassendiagramme für das Domain Model oder W3C CC/PP, Organigramme und Ortsmodelle für das Context Model. Für alle diese Modelle bietet UsiXML XML-basierte Repräsentationen. Alle Transformationen der Modelle werden durch Graph Grammatiken (vgl. [Roz97]) beschrieben, die auf diesen XML-basierten Repräsentationen definiert werden. Das dadurch definierte Regelwerk beschreibt dabei nicht nur die einmalige Transformation von einem Modell in das andere, sondern definiert, wie sich die Modelle zueinander verhalten. Wird ein Modell manipuliert, so können durch diese Regeln die Auswirkungen auf die generierte Benutzerschnittstelle ermittelt werden.

UsiXML setzt das in Abschnitt 2.2 vorgestellte Reference Framework vollständig um und spezifiziert für die dort beschriebenen Entwicklungsschritte die folgenden Modelle: Task-Model, Abstract und Concrete Interface. Die Final UI wird nicht durch UsiXML spezifiziert. Hier wird gefordert, dass Parser bereitgestellt werden, welche die CUI auf die tatsächliche Benutzerschnittstelle übersetzen.

Abbildung 4 veranschaulicht die unterschiedlichen Abstraktionsebenen und vermittelt einen Eindruck für die schrittweise Konkretisierung zwischen den Ebenen.

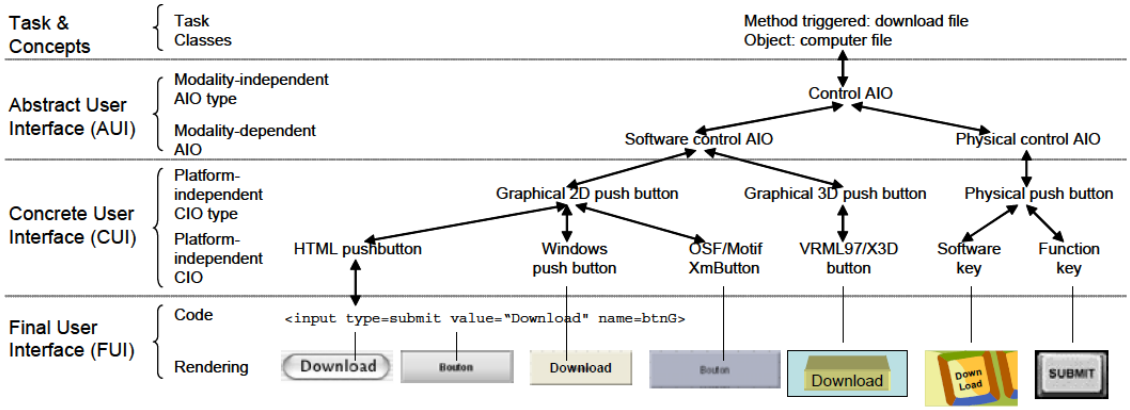


Abbildung 4: Transformation in den Ebenen des Reference Frameworks in UsiXML aus [LVM*05]

UsiXML wurde zum ersten Mal 2004 veröffentlicht und im Rahmen unterschiedlicher Forschungsprojekte weiter entwickelt: „Cameleon research project“ im Rahmen des European Fifth Framework Programme (FP5-2000-IST2); „SIMILAR network of excellence“ des European Sixth Framework Programme (FP6-2002-IST1-507609); und „HUMAN project“ (FP7-AAT-2007-RTD-1) des European 7th Framework Programme. Aktuell wird es im Rahmen des ITEA 2 (Eureka project 3674) mit dem Ziel weiter bearbeitet, UsiXML als Standard für die Beschreibung von Benutzerschnittstellen in Europa weiter zu etablieren.

2.3.3 UWE

UWE (UML-based Web Engineering, vgl. [Koc07], [KKZB08]) definiert eine Erweiterung der Unified Modelling Language (kurz UML, vgl. [Oes05]), die es ermöglicht, Web-Anwendungen zu spezifizieren. Als modellbasierter Ansatz definiert UWE eine Reihe von Modellen, welche unterschiedliche Aspekte einer Web-Anwendung modellieren. Für die Beschreibung der einzelnen Modelle werden ausschließlich UML-Konstrukte verwendet. Dabei wird die Semantik aller verwendeten UML-Modelle beibehalten. Die notwendigen Erweiterungen werden in Form von UML-Profilen realisiert (Light weight extensions, vgl. [Oes05]), so dass existierende Case-Tools, die UML unterstützen, für die Modellierung verwendet werden können.

UWE umfasst neben der Modellierungssprache auch Metamodelle, einen Entwicklungsprozess und Entwicklungswerkzeuge. In einem ersten Schritt werden die Kernmodelle präsentiert. Diese sind im Einzelnen:

Requirements Model, das Anforderungen an eine Web-Anwendung in Form von Anwendungsfällen beschreibt. Für die Modellierung werden Use Case- und Aktivitätsdiagramme verwendet.

Content Model, das Datenstrukturen definiert, die für die Anwendung relevant sind. Für die Modellierung werden UML-Klassendiagramme verwendet.

Process Flow Model, das ein Task-Model definiert (vgl. Abschnitt 5.1.4.2). In UWE wird das Process Flow Model verwendet, um Aktivitäten des Anwenders, aber auch des Systems zu erfassen. Die Dekomposition von Tasks in Sub-Tasks erfolgt dabei nicht in diesem Modell, sondern wird auf Basis des Requirements Models durchgeführt. Die Aktivitäten des Process Flow Models werden nicht weiter zerlegt. Für die Modellierung werden UML Aktivitätsdiagramme verwendet.

Navigation Model, das auf einer abstrakten Ebene die Interaktionen des Benutzers beschreibt. Grundlage für die Beschreibung sind die Datenstrukturen, welche aus dem Content Model heraus entwickelt werden. Das Navigation Model wird erzeugt, indem in einem ersten Schritt beschrieben wird, von welchen Datenstrukturen aus der Anwender andere Daten erreichen kann. In einem zweiten Schritt wird schließlich spezifiziert, auf welche Art von einem Datenobjekt zu einem anderen navigiert wird. Hierfür wurden die folgenden Strukturen definiert: „index“ (direkte Verknüpfung), „query“ (dynamische Liste), „menu“ (Liste) und „guided tour“ (Sequenz der anderen Strukturen). Für die Modellierung werden UML-Klassendiagramme verwendet.

Abbildung 5 visualisiert das Navigation Model, das bereits um Informationen des Process Flow Models erweitert wurde. Die „index“, „menu“ und „navigationClass“ Objekte werden aus dem Content Model ermittelt, während die „processClass“ Objekte aus dem Process Flow Model ermittelt werden. Durch die Nutzung beider Modelle (Navigation und Process Flow Model) ist es möglich, sowohl die Navigation über Datenstrukturen als auch die Aktivitäten im Bezug auf den Ablauf der Anwendung abzubilden.

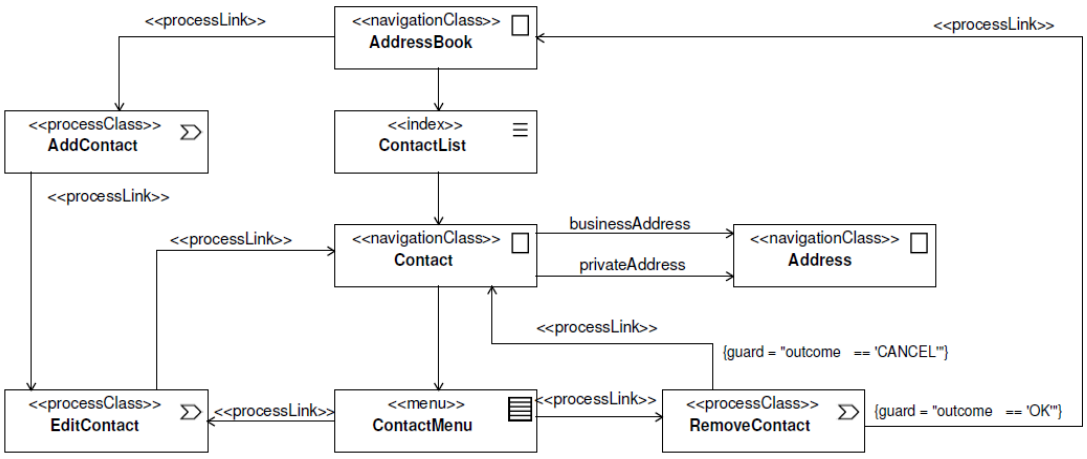


Abbildung 5: Erweitertes Navigation Model aus [RKKR*09]

Presentation Model, das auf einer abstrakten Ebene die Benutzerschnittstellen durch Stereotypen (Text, Button, Form, Link, Bild) beschreibt. Ermittelt wird das Presentation Model auf Basis des erweiterten Navigation Models. Für die Modellierung werden UML Klassendiagramme verwendet.

Abbildung 6 visualisiert solch eine Beschreibung einer Benutzerschnittstelle, die aus dem in Abbildung 5 dargestellten Navigation Model abgeleitet wurde.

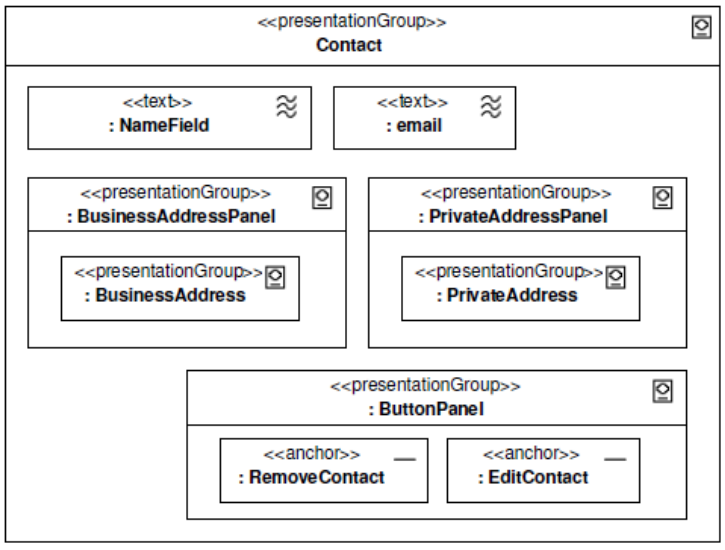


Abbildung 6: Presentation Model aus [RKKR*09]

Weitere Modelle wie beispielsweise ein User Model und auch ein Context Model wurden eingeführt, um zusätzliche Aspekte für die Modellierung zu erschließen. Diese können Auswirkungen auf die oben beschriebenen Modelle haben und zu strukturellen Änderungen oder Verfeinerungen in den Modellen führen.

Neben den Modellen beinhaltet UWE auch einen Entwicklungsprozess. Wie man bereits der Beschreibung der Modelle entnehmen kann, setzt UWE auf eine schrittweise Konkretisierung der Modelle bis hin zum Presentation Model. Kernelement dieses Konkretisierungsprozesses bilden dabei Transformationsregeln, die mit der ATLAS Transformation Language (vgl. [JK06]) beschrieben werden. ATLAS ist eine hybride (deklarativ und imperativ) Beschreibungssprache für die Transformation von Modellen. Werkzeuge, die ATLAS unterstützen, sind oftmals Teil von UML Case-Tools[3].

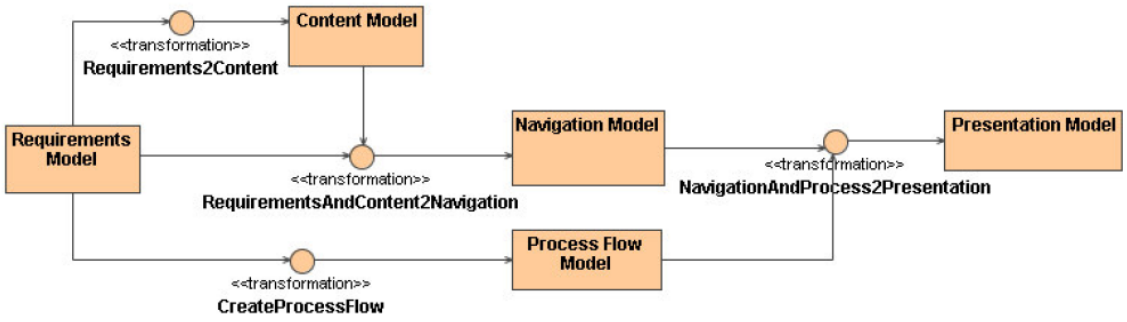


Abbildung 7: Generierungsprozess in UWE aus [KKK07]

Abbildung 7 visualisiert den Generierungsprozess in UWE. Die Knoten „transformation“ zeichnen dabei die Stellen im Prozess aus, in denen Transformationen eingesetzt werden, um ein neues Modell zu generieren. Dabei handelt es sich nur teilweise um ein automatisiertes Verfahren. Es ist vielmehr vorgesehen, dass automatisiert eine erste Version generiert wird, die durch den Modellierer verfeinert bzw. angepasst werden muss.

UWE bietet selbst keine direkte Unterstützung für die Generierung von Benutzerschnittstellen für unterschiedliche Plattformen oder Nutzer. Wie solche Konzepte in die Methodik zu integrieren sind, wird beispielsweise in [PLSZ*08] präsentiert. Hier wird eine Verknüpfung der RUX Methodik (Rich User Interface Experience Model, vgl. [LLPR*09]) mit UWE vorgestellt.

2.4 Single Layer Systeme

Die drei in den vorherigen Abschnitten vorgestellten Multi-Layer Ansätze für modellbasierte Systeme wurden ausgewählt, um den aktuellen Stand der Forschung wieder zu geben. In der Praxis haben sich solche Systeme aber noch nicht etablieren können.

Betrachtet man kommerzielle Werkzeuge für die Benutzerschnittstellengenerierung (bspw. Visual Studio von Microsoft, Eclipse der Eclipse Foundation, SunOne von Sun, usw.), so findet sich (wenn überhaupt) nur eine Abstraktionsebene auf der Benutzerschnittstellen spezifiziert werden. Um den aktuellen Stand

[3] ATLAS (kurz ALT genannt) wird beispielsweise von Eclipse unterstützt (http://www.eclipse.org/m2m/atl/)

kommerzieller Systeme zu beleuchten, wurden drei Benutzerschnittstellenspezifikationen ausgewählt und untersucht, welche Spezifikationsmethoden angeboten werden, aus denen heraus ausführbare Benutzerschnittstellen generiert werden.

Ausgewählt wurde hierzu als erstes die User Interface Markup Language (kurz UIML, vgl. [Pha00]). UIML wird als XML-basierte Metasprache bezeichnet, die es ermöglicht, Benutzerschnittstellen auf einer abstrakten Ebene zu beschreiben. UIML befindet sich in einem OASIS[4] Standardisierungsverfahren, um UIML als einheitliche Spezifikation von Benutzerschnittstellen zu etablieren.

Als zweite Spezifikationsmethode wurde die „XML User Interface Language“ (kurz XUL, vgl. [Dra05]) ausgewählt. Sie ist eine XML-basierte Beschreibungssprache, die ursprünglich im Rahmen des Mozilla-Projektes[5] entwickelt wurde, um die Portierbarkeit von Browser-Anwendungen zwischen unterschiedlichen Plattformen gewährleisten zu können.

Als Drittes wird XAML (Extensible Application Markup Language, vgl. [Joh05]) vorgestellt. XAML ist eine von Microsoft eingeführte XML-basierte Sprache, die im Rahmen der Windows Presentation Foundation[6]. für die Beschreibung von Benutzerschnittstellen verwendet wird. Sie ist integraler Bestandteil des Microsoft .NET-Frameworks.

Aus der Perspektive des Reference Frameworks nutzen kommerzielle Systeme für die Benutzerschnittstellengenerierung lediglich den letzen Schritt, nämlich die Generierung von Benutzerschnittstellen aus einer Spezifikation auf der Abstraktionsebene „Concrete User Interface“ heraus. Lediglich UIML bietet tatsächlich eine abstrakte Beschreibung (geräte- und plattformunabhängig) an, aus der aber direkt eine ausführbare Benutzerschnittstelle generiert wird, ohne dass eine „Concrete User Interface“-Abstraktionsschicht genutzt wird. Daher wird UIML ebenfalls im Rahmen der Single-Layer Systeme vorgestellt.

2.4.1 UIML

Anfangs (seit 1997) wurde die User Interface Markup Language (kurz UIML, vgl. [LRSV*09]) in Kooperation des Virginia Tech Research Centers[7] und der Firma Harmonia Inc.[8] entwickelt. Zielsetzung war es eine XML-basierte Metasprache bereitzustellen, die es Entwicklern ermöglicht, geräteunabhängig Benutzerschnittstellen zu beschreiben. Solch eine Beschreibung sollte mit Hilfe von geeigneten Parsern auf

[4] OASIS (Organization for the Advancement of Structured Information Standards), http://www.oasis-open.org

[5] https://developer.mozilla.org/en/XUL

[6] XAML wird wird im Rahmen der Windows Workflow Foundation auch zur Spezifikation von Workflows verwendet.

[7] http://www.vt.edu

[8] http://www.harmonia.com/

reale Endgeräte abgebildet werden. Heute wird UIML im Rahmen des OASIS[9] Standardisierungsverfahrens standardisiert. 2008 wurde die Version 4.0 von UIML als Community Draft (vgl. [HSLV*08]) veröffentlicht.

UIML setzt sich aus zwei zentralen Teilen zusammen: einerseits aus einer abstrakten Benutzerschnittstellenbeschreibung und andererseits aus einem Mapping-Vokabular, mit dem die abstrakte Beschreibung auf eine konkrete Realisierung einer Benutzerschnittstelle abgebildet werden kann.

Eine zentrale Fragestellung dabei ist, wie abstrakte Benutzerschnittstellen beschrieben werden können. Abbildung 8 veranschaulicht den Aufbau einer abstrakten Benutzerschnittstelle („Interface“ in Abbildung 8) in UIML. Verwendet wird eine Differenzierung in vier Bereiche: Structure, Style, Content und Behavior (vgl. [Pha00]).

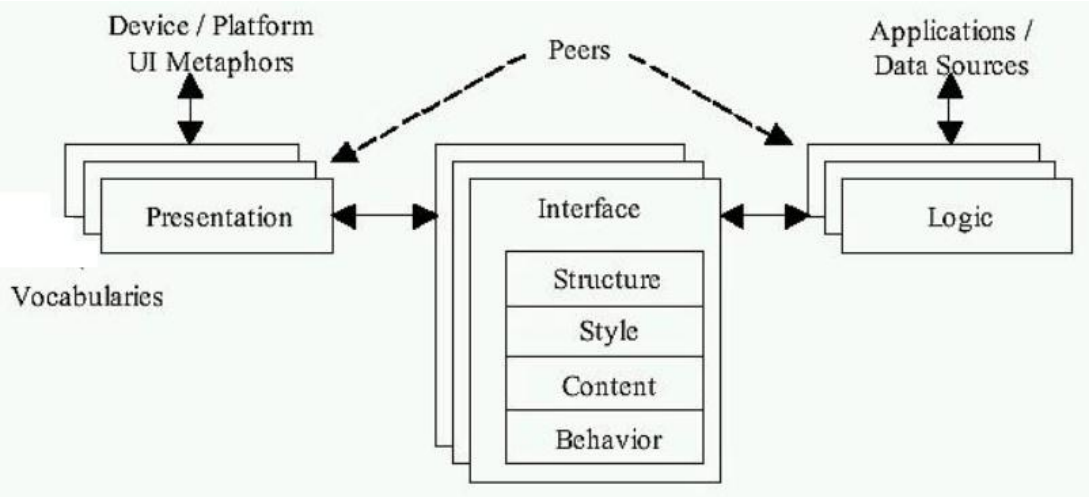


Abbildung 8: Der Aufbau einer UI-Beschreibung mit UIML aus [Pha00].

Structure beschreibt den Aufbau einer Benutzerschnittstelle. Dies sind Elemente einer Benutzerschnittstelle und deren Beziehungen untereinander. Solche Elemente sind in UIML durch „parts“ gegeben, die einen Teil einer Benutzerschnittstelle beschreiben. Ein „part“ kann beispielsweise ein Button oder eine Liste sein. Weiterhin sind „parts“ hierarchisch geordnet. So kann beispielsweise eine Liste ein „part“ sein, das aus einer Menge von Elementen (ebenfalls „parts“) besteht. In einem UIML-Dokument können unterschiedliche Structures existieren, so dass alternative Beschreibungen spezifiziert werden können. Erst im Generierungsschritt wird ausgewählt, welche Structure für das gewählte Endgerät am besten geeignet ist.

Style spezifiziert Eigenschaften (bspw. Button, unbeschriftet) wie ein „part“ präsentiert werden soll. Diese Eigenschaften werden jedem „part“ zugeordnet, der in der Structure definiert wurde. Weiterhin ist es möglich, unterschiedliche Styles für einen „part“ zu erstellen.

[9] OASIS (Organization for the Advancement of Structured Information Standards), http://www.oasis-open.org

Content beschreibt schließlich Inhalte, wie beispielsweise Texte, Grafiken, aber auch Audio-Files, die mit den „parts“ verknüpft werden können. Durch die Definition alternativer Content-Elemente ist es möglich, Internationalisierung oder das Anzeigen unterschiedlicher Inhalte (beispielweise für Experten oder Laien) zu realisieren.

Behavior spezifiziert schließlich das Verhalten der Benutzerschnittstelle. Verwendet wird ein ECA-Prinzip (Event/Condition/Action). UIML bietet dem Entwickler zwei Arten von Bedingungen an: eine Bedingung kann erfüllt werden, wenn ein Event eintritt, oder wenn ein Event eintritt und eine Bedingung erfüllt ist. UIML stellt vier Aktionstypen bereit. Sie können in interne und externe Aktionen unterschieden werden. Interne Aktionen verändern die Werte einer Benutzerschnittstelle, restrukturieren sie neu oder lösen ein Event aus. Externe Aktionen können Skripte, Programme oder Objekte aufrufen. Events werden beispielsweise durch das Drücken eines Buttons ausgelöst, oder durch andere Interaktionen mit der Benutzerschnittstelle.

UIML unterscheidet (dem MVC-Pattern entsprechend (vgl. [Mye98])) zwischen: View, das durch das Rendering des Interfaces (in Abbildung 8) auf ein bestimmtes Endgerät generiert wird (Presentation in Abbildung 8); dem Model, das durch das Interface (Structure, Style, Content, Behavior) beschrieben ist, und schließlich dem Control (Logic in Abbildung 8).

Mit „Peers“ werden in UIML die Komponenten Presentation und Logic (vgl. Abbildung 8) bezeichnet. Presentation definiert dabei für jedes Endgerät ein eigenes Vokabular, das Realisierungen von Elementen (Widgets) der Benutzerschnittstelle beschreibt. Über ein Mapping muss nun jedes „part“-Element (unter Berücksichtigung der angegebenen Eigenschaften) der abstrakten Benutzerschnittstelle auf das jeweilige Vokabular abgebildet werden. Das Mapping muss manuell erstellt werden. Der „Peer“ Logic beschreibt schließlich Zugriffe auf externe Datenquellen oder Applikationen, die Informationen für die Benutzerschnittstelle bereitstellen. Spezifiziert werden solche Zugriffe im Behavior, aber auch im Content Teil der abstrakten Benutzerschnittstellenspezifikation. Zusätzlich bietet UIML die Möglichkeit zur Definition von Templates. Diese beschreiben Teile einer Benutzerschnittstelle, die wiederverwendet werden können. Eine Spezifikation und die Definition der entsprechenden XML-Namensräume findet sich in [HSLV*08].

Für UIML sind bereits erste Versionen von Vokabularen und die notwendigen Parser (für Java, HTML, WML, VoiceXML, aber auch für PalmOS, C++ und .Net) verfügbar. Weiterhin existieren Werkzeuge, wie beispielsweise SketchiXML oder LiquidApps der Firma Harmonia (vgl. [LRSV*09]), welche UIML Spezifikationen generieren. Trotz der Standardisierungsbemühungen, des langen Entwicklungszeitraumes (über 10 Jahre) und der hohen Bekanntheit in der wissenschaftlichen Community, konnte sich UIML als Standard weder im wissenschaftlichen noch im wirtschaftlichen Bereich etablieren (vgl. [HA08]).

2.4.2 XUL

Die „XML User Interface Language“ (kurz XUL) ist eine XML-basierte Beschreibungssprache, die ursprünglich im Rahmen des Mozilla-Projektes[10] entwickelt wurde. Ziel der Entwicklung war es, mit XUL eine Beschreibungssprache für Anwendungen mit komplexen Benutzerschnittstellen zu bieten, die plattformübergreifend eingesetzt werden kann. Mozilla Anwendungen, wie beispielsweise Firefox und Thunderbird, wurden mit XUL erstellt. Eine mit XUL beschriebene Benutzerschnittstelle wird von einer Laufzeitumgebung (bspw. der Gecko Runtime Environment) interpretiert. Diese bietet eine Umgebung für alle Anwendungen, die mit XUL erstellt wurden. Alle Mozilla Produkte setzen solch eine Laufzeitumgebung ein. Dabei wird mit jedem Mozilla Browser die komplette Laufzeitumgebung bereitgestellt, die auch für andere Anwendungen genutzt werden kann. Mit XUL erstellte Anwendungen können sowohl Online- als auch Offline betrieben werden (vgl. [Joh05]).

Im Unterschied zu UIML, das die Benutzerschnittstelle auf einer abstrakten Ebene beschreibt, spezifiziert XUL die Benutzerschnittstelle direkt. Das bedeutet, dass Widgets und Fenster, die in XUL als Interface-Elemente bezeichnet werden, direkt durch die Laufzeitumgebung interpretiert und ausgeführt werden können. Die Spezifikation einer Benutzerschnittstelle ist somit direkt auf die Ziel-Plattform hin ausgerichtet. Mit XUL können aufgrund der existierenden Interface-Elemente nur Fenster-basierte, graphische Benutzerschnittstellen beschrieben werden.

XUL bietet eine Trennung zwischen der Präsentation und der Interaktionen auf der Benutzerschnittstelle und der Programmlogik. Zur Spezifikation der Interaktionen eines Benutzers mit der Benutzerschnittstelle werden Skripte (Java-Skript, vgl. [ECMAS]) verwendet. So kann beschrieben werden, wie sich Aktivitäten des Benutzers auf die Benutzerschnittstelle auswirken. Für die Beschreibung des Layouts und der gesamten Präsentation werden Cascading Style Sheets (kurz CSS, vgl. [CSS2]) verwendet. XUL bietet zusätzlich die Möglichkeit, sogenannte Skins zu definieren, die aus CSS und Bildern bestehen, um ein austauschbares „Look and Feel“ der Anwendung zu ermöglichen. Wesentliche Technologien von XUL sind (vgl. [Dra05]):

- „XPCOM/XPConnect (Cross Platform Component Object Model, Cross Plattform Connect) ermöglichen es, externe Bibliotheken aus den in XUL definierten Skripten heraus anzusprechen. Die Realisierung solcher Bibliotheken kann in unterschiedlichen Programmiersprachen erfolgen. Dadurch ist es möglich, neue Widgets zu definieren, die beispielsweise direkt mit externen Datenquellen verbunden werden.
- Die Beschreibung von Daten erfolgt über das Resource Description Framework (RDF, vgl. [BB04]), das in XUL eingebunden werden kann.

[10] https://developer.mozilla.org/en/XUL

- Extensible Bindings Language (XBL) ist eine eigene Auszeichnungssprache, die es ermöglicht, das Verhalten von Interface-Elementen von XUL zu beschreiben oder um existierende Interface-Elemente zu erweitern (Binding). Das Erweitern von Interface-Elementen umfasst dabei die Adaption der graphischen Repräsentation, das Hinzufügen von neuen Event-Handlern oder neuen Eigenschaften.
- Overlays bieten in XUL schließlich die Möglichkeit dynamisch XUL-Beschreibungen zu erweitern oder Teile einer XUL-Spezifikation zu überschreiben.

Bis auf RDF handelt es sich bei den genannten Technologien um spezifische Lösungen, die im Rahmen des Mozilla Projektes entwickelt wurden (vgl. [XUL10]).

XUL ist eine etablierte Sprache für die Beschreibung von Benutzerschnittstellen, die zusammen mit den Mozilla Web-Browsern eine weite Verbreitung erreicht hat. Für mobile Endgeräte bietet Mozilla einen eigenen „Fennec“ (Wüstenfuchs) genannten Web-Browser an. Dieser basiert ebenfalls auf XUL und bietet auch die oben genannten Erweiterungen. Durch die Änderungen im Aufbau der Benutzerschnittstelle, die sich durch das kleine Touch-Display und die fehlende Keyboard-Unterstützung ergeben haben, wurde die XUL Beschreibung auf nur wenige Elemente reduziert. Weiterhin sind spezifische Elemente für die mobile Nutzung hinzugekommen (vgl. [Moz10]).

Die Entwicklung von Fennec ist noch relativ neu. Erst 2009 veröffentlichte Mozilla den ersten Browser für Maemo[11] (ein Nokia Betriebssystem für Smart Phones). Ein erster Browser für Microsoft Windows Mobile 6[12] befindet sich noch in der Erprobung und war im März 2010 noch nicht veröffentlicht.

2.4.3 XAML

Mit XAML (Extensible Application Markup Language) führte Microsoft eine eigene XML-basierte Sprache für die Beschreibung von Benutzerschnittstellen ein. XAML[13] ist integraler Bestandteil der Windows Presentation Foundation, die für die graphische Darstellung von Benutzerschnittstellen im Rahmen des .NET Frameworks verantwortlich ist. Erstellt werden XAML Dokumente sowohl aus „Microsoft Visual Studio“ als auch mit „Microsoft Expression Blend“, das sich insbesondere an Designer von Benutzerschnittstellen richtet, die unabhängig von der Implementierung Benutzerschnittstellen erstellen können. XAML wird von Microsoft als eigener Standard im Rahmen des „Microsoft Open Specification Promise“[14] (OSP) verteilt und kann frei genutzt werden.

---

[11] http://qt.nokia.com/products/platform/maemo

[12] http://www.microsoft.com/windowsmobile

[13] XAML wird wird im Rahmen der Windows Workflow Foundation auch zur Spezifikation von Workflows verwendet.

[14] http://www.microsoft.com/interop/osp

Konzeptionell gleichen sich XAML und XUL. Auf den ersten Blick, scheint der Unterschied nur darin zu bestehen, dass der in XAML verwendete Namespace starke Anlehnungen an die Window.Forms[15] aufweist. Erwähnenswert im Rahmen von XAML ist die Zielsetzung von Microsoft, dass Entwickler, die XAML nutzen, keine Einschränkungen im Vergleich zur Entwicklung mit einer Programmiersprache haben sollen. Im Gegensatz zu XUL, das ursprünglich als Beschreibungssprache für einen Web-Browser entwickelt wurde, wurde XAML als allgemeine Beschreibungssprache für graphische Benutzerschnittstellen konzipiert. Dies zeigt sich beispielsweise durch die Möglichkeit auch komplexe Eigenschaften von Objekten in XML zu spezifizieren, so dass die Beschreibungsmöglichkeiten der Sprache erheblich erweitert werden. Wie bei XUL wird auch in XAML ein Event-Modell spezifiziert, und ein Mapping auf Funktionen, die diese Events verarbeiten (vgl. [Joh05]).

Eine weitere Neuerung führte Microsoft mit Silverlight (vgl. [DD08]) ein. Mit Silverlight bietet Microsoft die Implementierung eines browser- und plattformunabhängigen Client-seitigen Frameworks an. Silverlight lässt sich in drei Komponenten unterteilen: ein „Browser-Plugin“, ein “Presentation Framework“ und schließlich das „.NET Framework“. Abbildung 9 visualisiert diese Komponenten und benennt die verwendeten Technologien. Auf der untersten Ebene steht das Browser-Plugin. Wenn ein in HTML oder ASP.NET eingebettetes Objekt als Silverlight Anwendung erkannt wird, lädt das Plugin das dazugehörige „XAP-Package“, das den kompilierten Code der Anwendung enthält und führt dieses im Web-Browser aus (vgl. [DD08]).

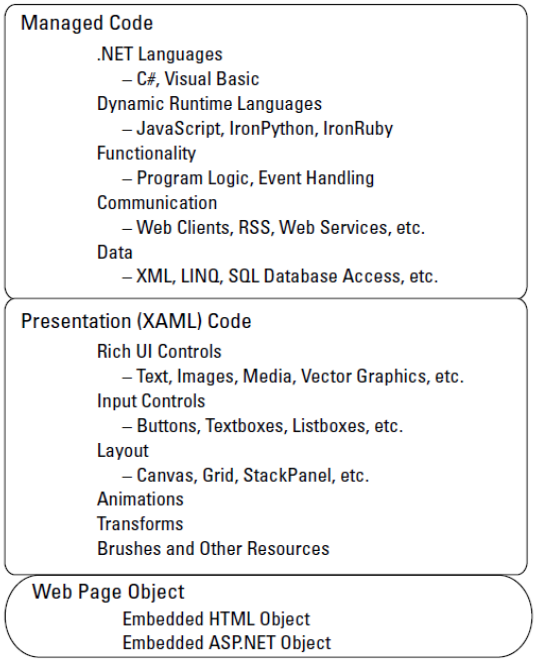


Abbildung 9: Silverlight Application Architecture Framework aus [DD08]

[15] Mit Windows.Forms ist der in .NET verwendete Namespace für die Beschreibung von Objekten der

Die mittlere Schicht beinhaltet das „Presentation Framework“. Zentraler Bestandteil sind dabei ein oder mehrere XAML Dokumente zur Beschreibung der gesamten Benutzerschnittstelle. Die gesamte Präsentationsschicht wird dabei durch XAML vollständig beschrieben.

Die oberste Schicht beschreibt schließlich den Zugang zu Datenbanken und Diensten, die mit der Benutzerschnittstelle verbunden sind.

Aktuell (März 2010) bekommt die Diskussion um XAML eine zusätzliche Bedeutung. Benutzerschnittstellen für das zukünftige Windows CE sollen nur noch auf Basis von Silverlight mit XAML beschrieben werden. Die noch nicht veröffentlichte nächste Version (Windows Mobile 7) sieht für die Entwicklung von Benutzerschnittstellen nur noch zwei Frameworks vor: Silverlight UI Framework und XNA UI Framework. Letzteres liefert ein Framework, das ausschließlich auf die Entwicklung interaktiver Spiele ausgelegt ist[16].

Mit XAML und Silverlight bietet Microsoft damit eine einheitliche XML-basierte Spezifikation von Benutzerschnittstellen an, die sowohl auf Desktop-PCs, mobilen Endgeräten und Fernsehern (X-Box-Live und dem Windows Media Center) ausgeführt werden kann.

2.5 Zusammenfassung und Ausblick

Modellbasierte Systeme für die Generierung von Benutzerschnittstellen sind ein bekanntes und sehr weit entwickeltes Forschungsfeld. Die Untersuchung der Forschungslandschaft, die im Rahmen der Multi-Layer Systeme (vgl. Abschnitt 2.3) durchgeführt wurde, zeigt, dass die aktuellen Ansätze viele innovative Lösungen bieten, die bereits eine hohe Reife erfahren haben. Auch die Konzeption und Entwicklung der notwendigen Werkzeugunterstützung ist in diesem Bereich sehr fortgeschritten und ermöglicht den praktischen Aufbau von Lösungen.

Auf der anderen Seite kann man in der Praxis nur wenige dieser Konzepte wiederfinden. Etabliert haben sich bisher tatsächlich nur Benutzerschnittstellenspezifikationen, die nah an der Implementierung liegen. Der Bedarf und auch die Notwendigkeit für neue Lösungen lässt sich aber aus den in Abschnitt 2.4 vorgestellten Systemen ableiten. Große Hersteller kommerzieller Software beginnen Benutzerschnittstellenspezifikationen zu verwenden und entwickeln Laufzeitumgebungen für unterschiedliche Plattformen und Endgeräte, die in der Lage sind diese auszuführen. Damit schaffen sie technische Grundlagen, die es ermöglichen, Benutzerschnittstellen flexibel (sei es nun für unterschiedliche Endgeräte, Plattformen oder Nutzer) generieren zu können.

Benutzerschnittstelle gemeint.

[16] vgl. Application Platform Overview for Windows Mobile Phone 7 (March 2010) (http://developer.windowsphone.com/windows-phone-7-series/)

Sei es nun die Komplexität der Modelle, der Mehraufwand für die Konzeption oder das Fehlen von Standards; eine abschließende Begründung warum es sich als schwierig gestaltet, die präsentierten Forschungsergebnisse in die Praxis zu übertragen, kann im Rahmen der hier erbrachten Betrachtungen nicht gegeben werden. Erkennbar ist aber, dass die Entwicklung von Benutzerschnittstellen als kreativer Akt verstanden wird, der den Einsatz von Menschen unabdingbar macht. Benutzerschnittstellen dürfen nicht mehr nur funktional sein, sondern müssen auch anderen Gesichtspunkten (Ausrichtung auf die Zielgruppe, ansprechende Gestaltung, Nutzungserlebnis) entsprechen. Eine weitere Feststellung ist, dass die Flexibilität und die Möglichkeiten, die ein Benutzerschnittstellen-Designer beispielsweise auf der Ebene der „Concrete User Interface“ hat, wesentlich höher ist als bei einer Spezifikation auf einem höheren Abstraktionslevel. Ein Benutzerschnittstellen-Designer kennt die Plattform und auch die Zielgruppe, für die er eine Benutzerschnittstelle entwirft, und wird dieses Wissen in einer Art umsetzen, die sich nur schwer in Modellen formalisieren lässt.

Der in dieser Arbeit entwickelte „Compositional Modeling“-Ansatz versucht nun, beide Welten (modellbasierte Ansätze und entwicklungsnahe Spezifikationsmethoden) näher zusammen zu bringen. Benutzerschnittstellenbausteine versprechen eine Entwicklung von Benutzerschnittstellen auf einer geräte-, plattform- und nutzerbezogenen Ebene. Die Erstellung dieser Bausteine erfolgt weiterhin manuell und ermöglicht Kreativität und Vielfalt in der Umsetzung. Benutzerschnittstellen-Designer können und sollen für die Realisierung Spezifikationsmethoden und Werkzeuge wählen, die ihnen die besten Möglichkeiten bieten. Die richtigen Bausteine für eine Anwendung zu identifizieren und diese Bausteine schließlich zu einer Benutzerschnittstelle zu komponieren ist schließlich die Aufgabenstellung, die im Rahmen dieser Arbeit behandelt wird.

Für den ersten Schritt werden modellbasierte Methodiken verwendet, um zu spezifizieren, was die Benutzerschnittstelle leisten soll und wie sie sich gegenüber dem Benutzer verhält. Der zweite Schritt, die Generierung durch Komposition, erfolgt schließlich auf Basis manuell erstellter Lösungsbausteine, die Wissen über die Realisierung kapseln, die nicht mehr Teil des modellbasierten Generierungsschrittes sein werden.

3 „Patientenorientierte telemedizinische Dienste“ als Anwendungsdomäne für das „Compositional Modeling“

„Patientenorientierte telemedizinische Dienste“ wurden für diese Arbeit als Anwendungsdomäne gewählt, anhand welcher der bausteinorientierte Ansatz des „Compositional Modeling“ für die Benutzerschnittstellengenerierung dargestellt wird. Im Rahmen dieses Kapitels wird diese Domäne eingeführt und motiviert, warum sich gerade hier Aufgabenstellungen identifizieren lassen, die neue Konzepte für die Benutzerschnittstellengenerierung erfordern. Dabei wird die Bedeutung von Service-orientierten Lösungen für die Telemedizin hervorgehoben. Schließlich wird der „Adipositas-Begleiter“ als konkrete Anwendung vorgestellt und damit das Anwendungsbeispiel eingeführt, an dem sich die Arbeit primär orientiert.

3.1 Motivation für patientenorientierte telemedizinische Dienste

Der demographische Wandel und der steigende Kostendruck im deutschen Gesundheitswesen erfordern neue, innovative Konzepte. Insbesondere telemedizinischen Diensten wird ein hohes Potenzial zur Reduktion von Kosten zugesprochen, da sie ein Verschieben von einer kostenintensiven, stationären Betreuung hin zu einer kostengünstigeren, ambulanten Betreuung ermöglichen.

Im Fokus dieser Arbeit stehen dabei patientenorientierte telemedizinische Dienste. Solche Dienste orientieren sich primär an Informations- und Unterstützungsbedürfnissen des Patienten in seinem Behandlungsverlauf mit der Zielsetzung, die Therapietreue (Compliance) des Patienten zu verbessern und ihn mittelfristig in die Lage zu versetzen, sich aktiv an seinem Gesundungsprozess zu beteiligen (Patient Empowerment).

Viele Reformen im Gesundheitswesen schreiben Patienten eine aktive Rolle als gut informierte Mitgestalter zu (vgl. [SW01]). Bekanntermaßen stellt der Patient selbst einen entscheidenden Faktor für den Heilungserfolg dar. So kann gezeigt werden, dass die Steigerung der Therapieerfolge und die Reduzierung der Kosten einhergehen mit einer besseren Einsicht der Patienten in die jeweilige Therapie und einem daraus resultierenden angemessenen Verhalten (Compliance). Die negativen Folgen und hohen Kosten der Nicht-Compliance benennen u.a. Sackett und Snow in [SS97]. Weiterhin zeigen Christensen und Griffiths in [CG00] am Beispiel des Internets, dass eine positive Korrelation zwischen der Informiertheit des Patienten und dem Heilerfolg besteht.

Dies findet auch Ausdruck in der Weiterentwicklung des Gesundheitsmarktes. Zusätzlich zur Primärversorgung hat sich in den letzten Jahren, ermöglicht durch die Gesundheitsreformen, ein zweiter und nunmehr auch ein dritter Gesundheitsmarkt etabliert. Während auf dem zweiten Gesundheitsmarkt das medizinische Personal noch der ausführende Akteur war, ist beim dritten Gesundheitsmarkt der Patient selber

die ausführende Person. Die Roland Berger Studie zum zweiten Gesundheitsmarkt beziffert diese neuen Gesundheitsmärkte mit einem Volumen von 60 Milliarden Euro im Jahr[17].

Als Anwendungsbeispiel wurde in dieser Arbeit der „Adipositas-Begleiter“ gewählt, der für die Behandlung chronischer Krankheiten eingesetzt wird. Er unterstützt zusammen mit minimalisierten klinischen Therapeutenkontakten stark übergewichtige Personen im täglichen Leben. In der Therapie erlernte Verhaltensweisen werden über Selbstkontrolltechniken (Planen, Erinnern, Protokollieren, Feedback) und psychologische Hilfen (Motivierung, Bewertung, Verstärkung, Spannung/Stressregulation) weiter eingeübt, um eine dauerhafte Lebensstiländerung zu erreichen.

Für die Realisierung solcher telemedizinischer Anwendungen sind zusätzliche Anforderungen zu berücksichtigen, die sich nicht nur aus der Perspektive des Datenschutzes, rechtlicher Rahmenbedingungen im Umgang mit Patientendaten und der Einbettung in die Gesundheitstelematik ergeben. Sie zeichnen sich insbesondere dadurch aus, dass sich die Zielgruppe der Patienten als höchst heterogen darstellt. Heterogen bedeutet dabei, dass Menschen unterschiedlichster Alters- und Bildungsstufen in der Lage sein müssen, telemedizinische Anwendungen zu bedienen und dabei oftmals auch mit körperlichen Behinderungen zu rechnen ist. Weiterhin muss weitgehend vom bisherigen Desktop-PC abstrahiert werden, da gerade in der Domäne der Telemedizin mobile oder in die Hausinfrastruktur integrierte IT-Systeme zunehmend Verwendung finden. Durch die Nutzung von intuitiven Interaktionsmedien wie beispielsweise Touch-Display, die Integration von Sensoren für Vitalparameter (EKG, Puls, Blutdruck, usw.) und die Nutzung bereits vorhandener häuslicher Infrastruktur (Fernseher mit Set-Top-Box) werden diese Systeme erstmals für die Patienten nutzbar.

Eine der wesentlichen Herausforderungen für telemedizinische Dienste wird es zukünftig sein, Anwendungen so anbieten zu können, dass sie von den Patienten akzeptiert werden. Nur wenn die Anwendungen auf die persönlichen Bedürfnisse des Benutzers ausgerichtet sind und sich in seine Umgebung integrieren, kann Telemedizin zu einem festen Bestandteil des Alltags werden.

Im Rahmen dieser Arbeit wird dabei der Begriff der „kontextorientierten Benutzerschnittstellen“ verwendet, um neben der Problematik unterschiedlicher Endgeräte (wie Desktop-PCs, Mobiltelefone, Set-Top-Boxen, Touch-Displays, usw.) auch den Patienten selbst und seine unmittelbare Umgebung mit einbeziehen zu können. Dabei müssen nicht nur die physischen Fähigkeiten des Patienten zur Nutzung des Endgerätes berücksichtigt werden, sondern auch wie weit der Patient mental in der Lage ist, dem Interaktionsablauf folgen zu können. Auch bei der Umgebung des Patienten sind nicht nur physische Merkmale wie das verfügbare Licht und der Umgebungslärm zu betrachten, sondern auch, ob und wie schützenswerte medizinische Daten dargestellt werden können. Alle diese Beispiele verdeutlichen, dass eine Änderung der

[17] Roland Berger Strategy Consultans (2007): „Roland Berger Studie zum Zweiten Gesundheitsmarkt: Chancen für Politik und Unternehmen aus (www.rolandberger.com Pressetext vom 5.7.2008)“.

Darstellung nur ein erster Schritt sein kann, und dass tiefgreifende Änderungen, die gegebenenfalls einen gänzlich anderen Prozess in der Benutzerinteraktion benötigen, notwendig sind.

Aus der Perspektive der Software-Entwicklung stellt sich diese Forderung als höchst problematisch dar. Benutzerschnittstellen werden oftmals als monolithischer Anwendungsbestandteil gesehen, der mit spezieller Werkzeugunterstützung (bspw. GUI-Builder) implementiert wird, und nur in einem gewissen Rahmen konfigurierbar ist. Dass es aber aufgrund des immensen Aufwandes nicht möglich ist, Benutzerschnittstellen für alle unterschiedlichen Variationen aus verfügbaren Endgeräten und Anforderungen der Nutzer zu erstellen, liegt dabei auf der Hand. Zwar ist insbesondere im Bereich der Telematik der Trend weg von monolithischen Anwendungen hin zu Service-orientierten Ansätzen erkennbar, doch bleibt dabei die Benutzerschnittstelle meist unbeachtet.

3.2 Notwendigkeit bausteinorientierter Konzepte für die Telemedizin

Aktuell kann auf eine Vielzahl von Projekten im Umfeld der Telemedizin (vgl. [Jäc09]) verwiesen werden. Allein in Nordrhein-Westfalen sind beispielsweise Vitaphone[18], DICOM Mail[19], Digitale Patientenbegleiter (vgl. [KLWK06]), SHL[20] und EPA.nrw[21] zu nennen, die in ganz unterschiedlichen Bereichen (Patienten Monitoring, PACS Systeme, Patientenakten, Patienteninformationssysteme, usw.) eingesetzt werden. Neben diesen Projekten wird auch intensiv an Grundlagen und Infrastrukturen für telemedizinische Dienste gearbeitet.

Trotz der dargestellten Notwendigkeit telemedizinische Dienstleistungen für das Gesundheitswesen und Projekte in diesem Bereich zu entwickeln, sind nur wenige etablierte Lösungen auf dem Markt erfolgreich. Es zeigt sich, dass Unternehmen, die telemedizinische Dienstleistungen auf den Markt bringen wollen, erheblichen Hürden gegenüberstehen. Dies ist einerseits auf mangelndes Wissen über Prozesse im Gesundheitswesen zurück zu führen, anderseits müssen auch hohe Vorinvestitionen (Aufbau von technologischem und fachlichem Know-how) geleistet werden; schließlich besteht Unsicherheit darüber, welchen gesetzlichen Vorschriften und Empfehlungen telemedizinische Anwendungen genügen müssen.

Vor diesem Hintergrund lassen sich eine Reihe von Aktivitäten identifizieren, mit welchen versucht wird, diese Hemmschwellen durch die Einführung von einheitlichen Infrastrukturen, Standards und insbesondere Service-orientierten Ansätzen in der Telemedizin zu verringern. So stellt die Telematikinfrastruktur eines der größten Projekte der Bundesregierung im Gesundheitswesen dar. Ziel ist es, eine einheitliche

[18] Vitaphone GmbH, http://www.vitaphone.de

[19] JiveX DICOM MAIL der Firma VISUS Technology Transfer GmbH, http://www.visus.com

[20] SHL Telemedizin GmbH, http://www.shl-telemedicine.de

[21] Projekt im Rahmen der Initiative eGesundheit.nrw des Ministerium für Arbeit, Gesundheit und Soziales des Landes NRW

und standardisierte Infrastruktur zwischen den Akteuren (Arztpraxen, Krankenhäusern, Apotheken und Krankenkassen) im Gesundheitswesen zu etablieren. Teil dieser Infrastruktur sind die elektronische Gesundheitskarte (eGK) und der Heilberufeausweis (HBA), und in einem ersten Schritt Anwendungen wie der Versichertendaten-Dienst oder das elektronische Rezept. Die Gesundheitstelematik stellt aber mit dem Konzept der Mehrwertdienste eine wichtige Schnittstelle bereit, um auch externe Dienstleister über die Telematikinfrastruktur anbinden zu können. Aktuell sind vier Typen von Mehrwertdiensten vorgesehen, die durch die Art der Kommunikation und deren Sicherheitsprofile beschrieben werden (vgl. [NDW06]).

Parallel zu diesen bundesweiten Aktivitäten wird bereits versucht, Märkte für telemedizinische Dienstleistungen zu etablieren. So stellt bereits heute die Telemedizin24[22] ein Portal bereit, das als Marktplatz für Diensteanbieter und –nutzer fungiert und zukünftig auch notwendige Vertrauensverhältnisse (bspw. Zertifizierung von Diensten & Dienstleistern) und Abrechnungskomponenten bereitstellen wird.

In Initiativen, wie beispielsweise den vom Ministerium für Arbeit, Gesundheit und Soziales des Landes NRW geförderten Projekt „Realisierung und Evaluation eines Repositories für patientenorientierte telemedizinische Dienste“[23], arbeiten Forschungsinstitute zusammen mit Partnern aus dem Mittelstand und medizinischen Einrichtungen zusammen. Ziel ist es, das fachliche Wissen über die Realisierung von telemedizinischen Diensten zu formalisieren und in Form eines Repositories (Software-Komponenten) einem breiten Anwenderkreis zur Verfügung zu stellen. Das Repository versteht sich dabei als ein Baukasten für telemedizinische Dienste, der im Rahmen einer vorwettbewerblichen Aktivität entwickelt wird und zukünftig helfen kann, Barrieren für die Einführung neuer telemedizinischer Dienstleistungen zu reduzieren. Es ermöglicht insbesondere auch kleinen und mittelständischen Unternehmen, nicht nur auf fachliches und technisches Know-how zurück zu greifen, sondern insbesondere auch existierende Implementierungen zu nutzen, um innovative telemedizinische Dienste auf den Markt bringen zu können. Das Repository selbst kann als ein weiterer Markt aufgefasst werden, über das Anbieter von telemedizinischen Dienstleistungen ihrerseits Softwarekomponenten anbieten können, um damit ihr Angebot zu erweitern. Die als „Services“ realisierten Bausteine können dabei auf unterschiedlichen Plattformen eingesetzt werden, so dass die Projektergebnisse auch anderen Initiativen und Projekten zur Verfügung gestellt werden.

[22] http://www.telemedizin24.de

[23] Siegerprojekt des vom Ministerium für Arbeit, Gesundheit und Soziales geförderten Wettbewerbs Med in NRW des Landes Nordrhein-Westfalen.

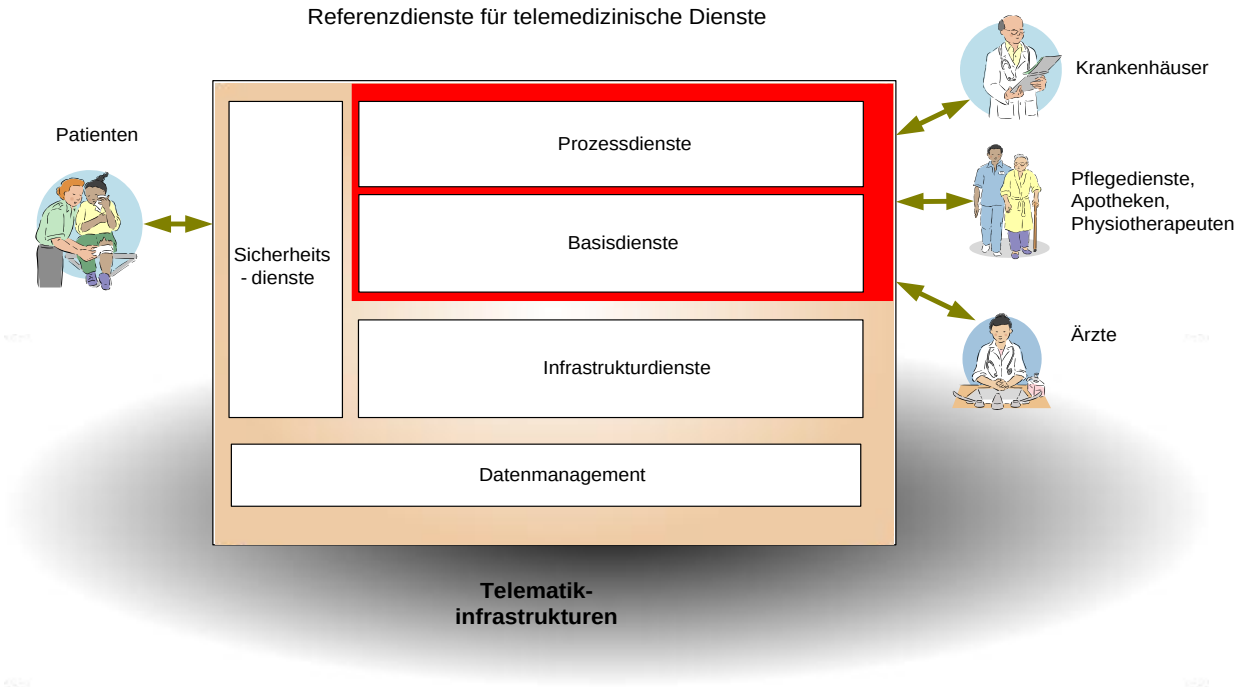


Abbildung 10: Diensteorientierte Architektur für telemedizinische Anwendungen

In Abbildung 10 wird der Baustein-Charakter des Repositories illustriert, der hier stellvertretend für ähnliche Ansätze vorgestellt wird. Die beiden dunkel umrandeten Rechtecke (Basisdienste und Prozessdienste) bilden die Schwerpunkte der Arbeiten für das Repository: die „Basisdienste“ kapseln grundlegende Funktionalitäten für telemedizinische Dienste ohne direkten fachlichen Bezug; die „Prozessdienste“ hingegen umfassen bereits spezifisches fachliches und insbesondere an den Prozessen im Gesundheitswesen orientiertes Wissen.

Diesen Diensten werden softwaretechnische Basisimplementierungen im Hinblick auf Sicherheitstechnik, Infrastruktur und Datenmanagement an die Seite gestellt, die für eine Service-orientierte Architektur benötigt werden. Diese drei Komponenten beschreiben schließlich die notwendigen Schnittstellen, die zu schaffen sind, um das Repository in anderen Projekten integrieren zu können.

Diesen Initiativen liegt dieselbe Zielsetzung zu Grunde, nämlich im Bereich Telemedizin konsequent durch den Einsatz von offenen Standards, Infrastrukturen, Service-orientierten Architekturen und schließlich dem Aufbau von Service-Märkten bausteinorientierte Softwareentwicklung zu ermöglichen. Getrieben durch die Gesundheitstelematik und Initiativen wie die elektronische Fallakte[24] etablieren sich hier wie in kaum einer anderen Anwendungsdomäne offene Web-Service-Standards.

[24] Elektronische Fallakten ermöglichen einen sicheren, datenschutzkonformen Austausch von medizinischen Daten in Versorgungsnetzen. Die von privaten, öffentlichen und frei-gemeinnützigen Kliniken getragenen Spezifikationen sind frei verfügbar. (vgl. www.fallakte.de)

Es ist aber zu berücksichtigen, dass die bisherigen Entwicklungen die Erstellung der Benutzerschnittstellen weitgehend ausklammern. Die Bereitstellung von Konzepten und Werkzeugen, die diese Lücke im Entwicklungsprozess schließen, ist eine konsequente Weiterführung der bisherigen Entwicklungsstrategie. Verschärft wird die Situation, wenn zusätzlich kontextorientierte Benutzerschnittstellen angeboten werden müssen, wie im Bereich der „patientenorientierten telemedizinischen Anwendungen“ gefordert wird.

3.3 Der Adipositas-Begleiter als Praxisbeispiel für den Generierungsprozess

Als Anwendungsbeispiel, an welchem die Generierung von Benutzerschnittstellen exemplarisch durchgeführt wird, wurde der Adipositas-Begleiter ausgewählt. Der Adipositas-Begleiter beschreibt ein informationslogistisches telemedizinisches System, das in der Nachsorgephase (in häuslicher Umgebung) einer stationären Adipositas-Therapie eingesetzt wird. In diesem Abschnitt wird der Adipositas-Begleiter als Anwendungsdomäne identifiziert, in welcher der in dieser Arbeit entwickelte „Compositional Modeling“-Ansatz zur Generierung von Benutzerschnittstellen eingesetzt werden soll. In einem ersten Schritt wird die Zielsetzung und Ausgangslage des Adipositas-Begleiters kurz umrissen, wobei eine detaillierte Darstellung der Konzepte, Lösungsansätze und Technologien bereits in [KLWK06] und [KK08] veröffentlicht wurde.

Entwickelt wurde der Adipositas-Begleiter in einer Kooperation des Fraunhofer Institutes für Software- und Systemtechnik (kurz ISST) und der Gelderland-Klinik. Zentrale Zielsetzung des Projektes ist die Verbesserung der Umsetzung und die Integration von Verhaltensänderungen in Alltag und Beruf, die durch eine stationäre rehabilitative Behandlung initiiert wurden. Zielgruppe ist die kostenintensive und hochgesundheitsgefährdete bzw. gesundheitsbeeinträchtigte Störungsgruppe der Patienten mit schwerer und extremer Adipositas (BMI >35).

Im Rahmen der 15-jährigen Erfahrung der Gelderland-Klinik in der stationären psychosomatischen Behandlung schwerer Adipositas hat sich gezeigt, dass sich die mit erprobten spezifischen Behandlungsprogrammen stationär erzielten positiven Erfolge (Gewichtsreduktion, Verhaltens- und Einstellungsveränderungen) längerfristig nicht aufrecht erhalten lassen. Trotz anfänglich hoher Motivation sind die Patienten im Alltag überfordert und geben erlernte Verhaltensweisen schnell wieder auf.

Aufgabe des Adipositas-Begleiters ist es, die positiven Veränderungen, die in der stationären Therapie erfolgreich initiiert wurden, in den Alltag des Patienten zu übertragen. Hierzu erhalten die Patienten nach Beendigung der stationären Therapie Zugriff auf telemedizinische Dienste, die sie dabei unterstützen, die stationär erworbenen Verhaltensweisen in das häusliche Umfeld zu übertragen.

Aus Sicht der Therapeuten zielt der Adipositas-Begleiter auf die folgenden Veränderungsfaktoren (vgl. [KLWK06]) ab:

1. Aktive Einbeziehung des Patienten in den Veränderungsprozess (Selbstmanagement). Der Patient ist aktiver Gestalter seiner Veränderungsprozesse. Er nimmt Veränderungen unmittelbar wahr und wird bei deren Bewertung unterstützt, um vorab definierte Ziele zu erreichen.
2. Motivation (Empowerment) sichern. Über den Adipositas-Begleiter hat der Patient Zugang zu Ansporn, Lob und Verstärkern, außerdem werden Handlungsalternativen aufgezeigt.
3. Bedarfsgerechte, jederzeit verfügbare, kleinschrittige, mobile, flexible und individuelle Unterstützung des Patienten in Alltag und Beruf. Die Informationen und Unterstützungen werden durch das mobile Gerät unmittelbar bereitgestellt, wenn der Patient sie benötigt. Dies ermöglicht Affekt- und Spannungsregulation bei Gefahr von Kontrollverlusten.
4. Fortführung der stationären Therapiebestandteile (Ernährung, Bewegung, Stressverarbeitung, Affekt- und Spannungsregulation) in der Nachsorgephase, Aufrechterhaltung der vertrauten Beziehung zum Reha-Team bei minimalem Personal- und somit Kostenaufwand.

Im Rahmen der seit 2003 laufenden Kooperation wurden basierend auf diesen Zielsetzungen in einem multidisziplinären Team bestehend aus Psychologen, Ernährungs-, Bewegungstherapeuten und IT-Experten telemedizinische Dienstleistungen konzipiert. Aufgegriffen wurden hierzu primär die Abläufe und Konzepte der stationären Therapie, die durch den telemedizinisch vermittelten Kontakt zu Therapeuten und der (sich im stationären Aufenthalt entwickelten) Therapiegruppe erweitert wurden. Eine Übersicht der entwickelten Kategorien und Dienstleistungen wird in Abbildung 11 dargestellt. Die drei mittleren Säulen Affektregulation, Ernährung und Bewegung bilden die zentralen Therapiebereiche der stationären Behandlung ab.

Im Bereich „Affektregulation“ trainiert der Patient, sich selbst kritisch zu hinterfragen, sein erfolgreiches Handeln positiv zu erfahren und sich in Stresssituationen eine emotionale Distanz erarbeiten zu können. Erweitert wird diese psychologische Begleitung durch Therapiekonzepte aus dem Bereich „Bewegung“ und „Ernährung“, in denen der Betroffene ein neues Verhalten erlernt hat und es in seinen eigenen Alltag übernehmen will. Diese drei Bereiche kategorisieren telemedizinische Dienstleistungen und dienen den Patienten gleichzeitig als Anhaltspunkt Dienstleistungen wieder zu finden.

Die Bereiche „Kritische Situationen“ und „Mein Digi“ verstehen sich als Sammlung von Diensten. Der erste Bereich liefert Hilfestellungen für schwierige Situationen, indem Erinnerungen an in der Therapie Erlerntes angeboten werden. Eine besondere Form der kritischen Situationen ist der Umgang mit Essdruck oder auftauchenden negativen Emotionen, die die Gefahr beinhalten, von Essen oder Essattacken „überwältigt“ zu werden. Hier bietet der „Notfallkoffer“ unmittelbare Hilfen zur Distanzierung, Ablenkung oder Umlenkung. „Mein Digi“ ermöglicht eine personalisierte Sicht auf alle Themenbereiche, persönliche Termine und Leistungen sowie einen Zugang zu den Protokollierungsdiensten.

Eine weitere Eigenschaft der über den Adipositas-Begleiter angebotenen telemedizinischen Dienste ist es, dass sie dem Nutzer im Laufe der Zeit zunehmend die Möglichkeit geben sollen, selbstständig aktiv zu

werden. So kann beispielsweise der Nutzer im Menü „Bewegung“ anfangs nur auf vordefinierte Aktivitäten zurückgreifen und erst später selbstständig Aktivitäten hinzufügen und dadurch das System erweitern.

In verschiedenen Foren können die Patienten untereinander oder mit den Betreuern in Kontakt treten.

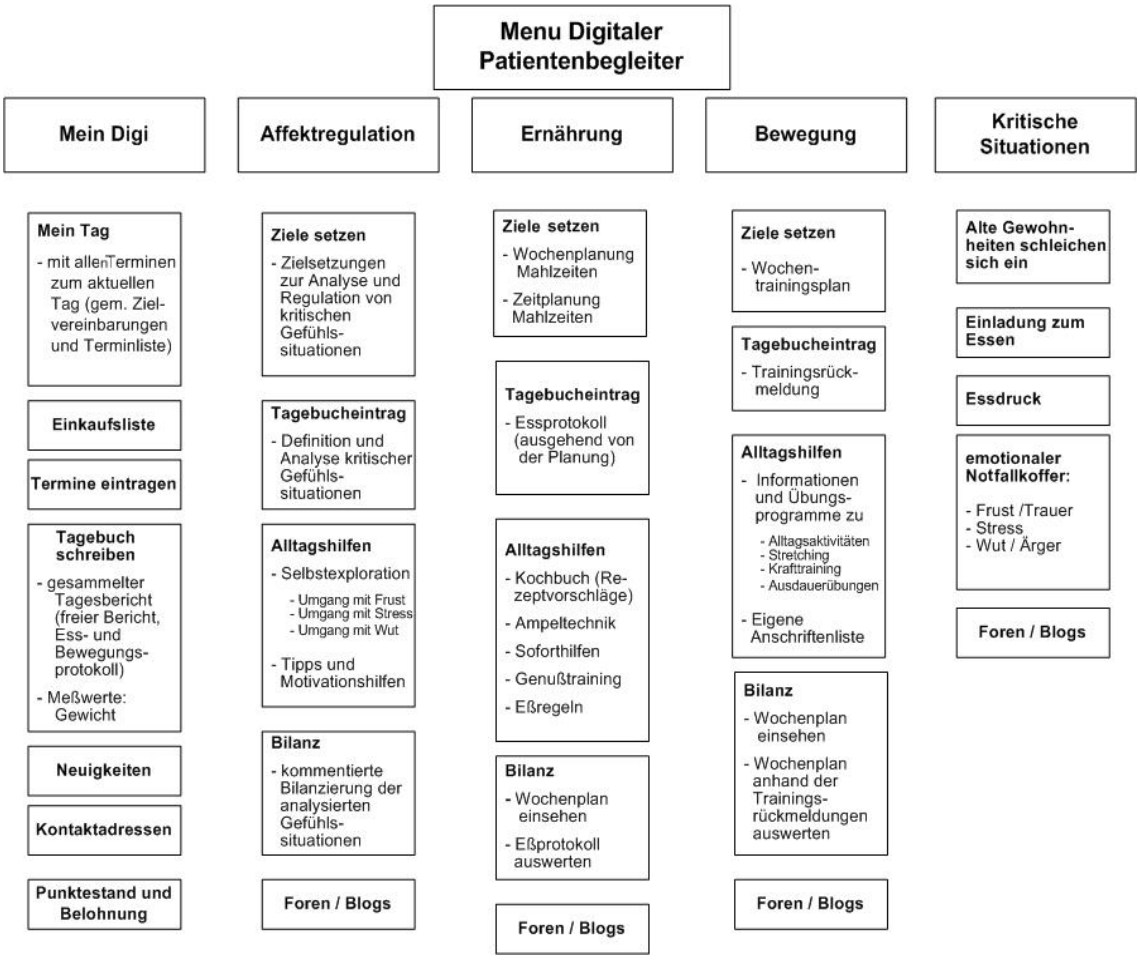


Abbildung 11: Dienstekategorien des Adipositas-Begleiters (in Anlehnung an [KLWK06])

Bei all diesen Diensten ist natürlich zu ergänzen, dass der Adipositas-Begleiter auch eine Schnittstelle für Therapeuten (Psychologen, Ernährungs- und Bewegungsexperten) bieten muss. Über diese Schnittstelle haben sie die Möglichkeit auf Basisdaten, Gewichtsverläufe, Zieldefinitionen, Nutzungshäufigkeit und vieles mehr zuzugreifen und auch zu reagieren. So besteht die Möglichkeit, dass Therapeuten gezielt Patienten ansprechen, sie motivieren oder intervenieren.

Realisiert und evaluiert wurde der Adipositas-Begleiter in einem ersten Schritt über mobile Endgeräte (PDAs und Smart Phones). Auf diese Schnittstelle hin wurden telemedizinische Dienstleistungen konzipiert, realisiert und zusammen mit Patienten evaluiert. Ein Ergebniss der Evaluation ist, dass nur ein Teil

der Zielgruppe auch wirklich affin für mobile Endgeräte ist. Oftmals wurde der Wunsch formuliert auch den heimischen PC einzusetzen oder es wurde gänzlich der Umgang mit Computern in Frage gestellt.

Zukünftig ist der Einsatz weiterer Engeräte geplant. Hier sollen insbesondere der Heim-PC und spezielle telemedizinische Endgeräte, die beispielsweise über ein Touch-Display zu bedienen sind, eingesetzt werden, um einen größeren Anwenderkreis bedienen zu können. Ein viel versprechendes Endgerät, das die Verbreitung telemedizinischer Anwendungen erleichtern könnte, kommt aus dem Bereich des AALs (Ambient Assisted Livings). Hier werden Set-Top-Boxen eingesetzt, die an den heimischen Fernseher angeschlossen werden und so Interaktionen über die Fernbedienung ermöglichen.

Diese Endgeräteproblematik ist dabei nur einer der Gründe, warum dieser Anwendungsbereich für den „Compositional Modeling"-Ansatz besonders geeignet ist. So haben Erfahrungen gezeigt, dass der potentielle Anwenderkreis der Adipositas-Patienten sehr breit gestreut ist und von Jugendlichen bis hin zu älteren Menschen reicht. Im besonderen Maße unterscheidet sich die Vorbildung der Anwender in der Nutzung von IT-Systemen, so dass eine Adaption der Benutzerschnittstelle absolut notwendig ist. So wurde beispielsweise beim Einsatz des mobilen Endgerätes kritisiert, dass ein hohes Maß an Feinmotorik nötig ist.

Die Menge an unterschiedlichen Diensten ist in diesem Bereich als sehr hoch einzuschätzen. Abbildung 11 vermittelt hier einen ersten Eindruck. Die Anzahl an verfügbaren Diensten muss weiterhin kontinuierlich erweitert werden, um den Adipositas-Begleiter auch über einen längeren Zeitraum hinweg attraktiv zu gestalten.

Bei der Benutzerschnittstelle ist weiterhin zu berücksichtigen, dass viele Elemente der Dienste wie beispielsweise Tagebücher in unterschiedlichen Kontexten (beispielsweise Bewegung und Ernähung) verwendet werden, so dass ein bausteinorientierter Ansatz besonders günstig erscheint. Abbildung 11 veranschaulicht diesen Sachverhalt, indem in den drei Therapiebereichen („Affektregulation", „Ernährung" und „Bewegung") die identische Strukturierung der Dienste in „Ziele setzen", „Tagebucheintrag" und „Bilanz" gewählt wurde.

Die Flexibilität der Dienste in Bezug auf die Realisierung der Benutzerschnittstelle spielt ebenfalls eine wichtige Rolle. So erfordert beispielsweise die Anforderung, dass die Patienten zunehmend selbstständig aktiv werden können, kontinuierliche Modifikationen der Benutzerschnittstelle. So soll der Patienten in der ersten Therapiephase nur auf vordefinierte Inhalte zugreifen können, und erst nach einiger Zeit auch die Möglichkeit erhalten, selbst Kategorien zu definieren. In der Endphase der Therapie soll er schließlich eigene Inhalte auch anderen Patienten bereitstellen können. Solch ein Vorgehen wurde von den Therapeuten als wichtig eingestuft, um einerseits die Einarbeitung in die Nutzung der Dienste zu erleichtern, andererseits aber auch eine positive Entwicklung in Richtung Selbstständigkeit greifbar darzustellen.

Die bisher genannten Eigenschaften zeigen, dass die Anwendungsdomäne des Adipositas-Begleiters ein flexibles Konzept zur effizienten Generierung von Benutzerschnittstellen benötigt. Weiterhin ist positiv zu bewerten, dass bereits Benutzerschnittstellen (für die Realisierung auf Basis von PDAs) existieren,

welche sich als eine erste Vorlage für den Aufbau einer Wissensbasis anbieten. Eine weitere Eigenschaft dieser Anwendungsdomäne ist ein gemeinsames Verständnis der Anwender für die angebotenen telemedizinischen Dienstleistungen. Jeder Anwender war auch stationärer Patient der Gelderland-Klinik und ist somit vertraut mit den Therapiekonzepten und den Zielsetzungen, die mit den Anwendungen verfolgt werden.

4 Blueprint für das Compositional Modeling

Im Fokus der vorliegenden Arbeit steht die Fragestellung, wie Benutzerschnittstellen für unterschiedliche Endgeräte, Anwender und auch Umgebungen der Nutzung erstellt werden können. Der gewählte Ansatz, um dieser Fragestellung zu begegnen, sieht vor, die Generierung von Benutzerschnittstellen weitgehend automatisiert ablaufen zu lassen. Vorraussetzung dafür ist, dass Modelle zur Verfügung stehen, mit denen beschrieben wird, welchen Anforderungen die zu generierende Benutzerschnittstelle gerecht werden muss und was die Benutzerschnittstelle schließlich leisten soll. Der Grundgedanke des „Compositional Modeling“ konkretisiert diesen Ansatz dahingehend, dass die Generierung von Benutzerschnittstellen durch die Komposition existierender Bausteine realisiert wird, welche in einem Baukasten bereitgestellt werden.

Innerhalb dieses Kapitels wird ein Blueprint aufgebaut, wie der „Compositional Modeling“-Ansatz auf die Generierung von Benutzerschnittstellen zu übertragen ist. Hierzu müssen der Generierung vorlaufend Modelle erstellt, Werkzeug bereitgestellt und insbesondere auch um dem „Compositional Modeling“ gerecht zu werden, wieder verwendbare Bausteine definiert werden. Im Rahmen dieses Kapitels wird daher in einem weiteren Schritt der gesamte „Entwicklungsprozess“ betrachtet, der das „Compositional Modeling“ in einen Softwareentwicklungsprozess einbettet. Diese Betrachtung soll insbesondere die Fragestellung beantworten, wie die erarbeiteten Methoden für die Softwareentwicklung nutzbar gemacht werden und welche Rollen (Designer, Modellierer, SW-Entwickler) an der Entwicklung beteiligt sind.

4.1 Nutzung des „Compositional Modeling“ für die Benutzerschnittstellengenerierung

Das Leitmotiv für die Generierung von Benutzerschnittstellen ist das „Compositional Modeling“ nach Falkenhainer und Forbus (vgl. [FF91]). Der Begriff „Compositional Modeling“ steht dabei für den verwendeten Ansatz, nämlich die Komposition zur Modellgenerierung zu nutzen.

Zur Veranschaulichung des Ansatzes wird das Einführungsbeispiel aus [FF91] für das „Compositional Modeling“ in der Anwendungsdomäne der Thermodynamik herangezogen. Abbildung 12 visualisiert das Beispiel, in welchem ein Brenner Wasser erhitzt und dadurch eine Turbine antreibt. Links in Abbildung 12 wird der Kompositionsprozess beschrieben, während auf der rechten Seite die drei Bestandteile des Ansatzes visualisiert sind. Im Einzelnen sind dies:

- Szenariobeschreibung (in Abbildung 12 Scenario Description genannt), welches eine Beschreibung der physischen Struktur und des Verhaltens der Elemente beinhaltet.
- Domain Theory, welche eine Bibliothek an unterschiedlichen Modellbausteinen umfasst, aus denen das Zielmodell durch Komposition generiert werden soll.
- Query, welche Anforderungen in Form einer Fragestellung formuliert, auf die das Zielmodell bestmöglich antworten liefern soll.

Das Ergebnis des Kompositionsprozesses wird „Scenario Model“ genannt und beschreibt ein Modell, das aus Elementen der „Domain Theory“ komponiert wurde und das in der Lage ist, die gegebene Query bestmöglich zu bedienen.

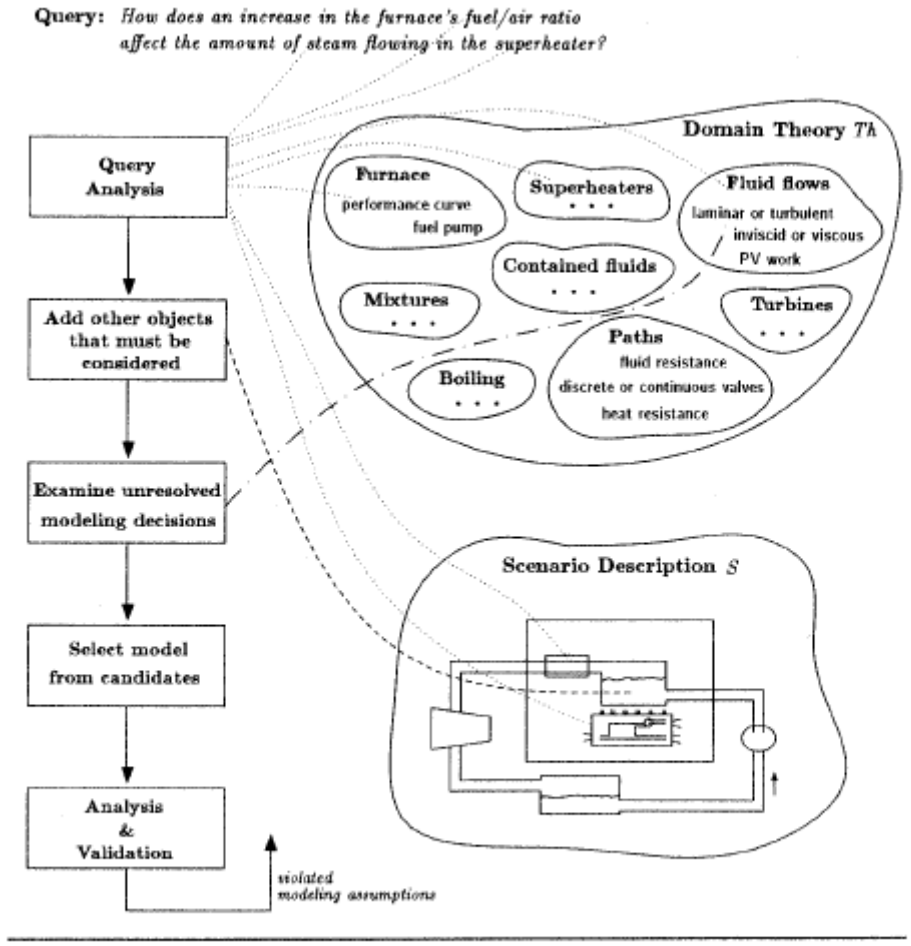


Fig. 2. Overview of the model composition process.

Abbildung 12: Kompositionsprozess nach Falkenhainer und Forbus (Abbildung aus [FF91])

Der in Abbildung 12 dargestellte Kompositionsprozess sieht die Analyse der Query (Schritt 1) und der Scenario Description (Schritt 2) vor, um relevante Elemente der „Domain Theory“ zu ermitteln. Zielsetzung ist es, für die gesamte Scenario Description eine vollständige und auf die gegebene Query hin optimierte Abdeckung durch Fragmente der „Domain Theory“ zu erhalten. Schritt 3 des Kompositionsprozesses bezieht sich dabei auf Restriktionen der Anwendungsdomäne, der ausgewählten Elemente der „Domain Theory“ und der Anforderungen der Scenario Description. Die letzten beiden Schritte veranschaulichen den zyklischen Entwicklungsprozess, in dem Modelle durch Komposition generiert, analysiert und evaluiert wurden und schließlich erweitert durch Informationen erneut in den Prozess gegeben werden. Auf eine genauere Betrachtung der Ansätze zur Komposition von Fragmenten und der Ermittlung von passenden Fragmenten wird an dieser Stelle verzichtet, da eine direkte Übertragung auf die Benutzer-

schnittstellengenerierung nicht realisierbar ist. Die Methoden des „Compositional Modeling“, wie sie in [FF91] vorgestellt wurden, basieren auf Prämissen[25], die sich nicht direkt auf Benutzerschnittstellen abbilden lassen.

Die Übertragung der Idee des „Compositional Modeling“ auf die Generierung von Benutzerschnittstellen erfordert in einem ersten Schritt die Definition passender Modelle, die analog zur „Domain Theory“, „Scenario Description“ und „Query“ verwendet werden können. Darauf aufbauend können in einem zweiten Schritt Methoden, Bibliotheken und Werkzeuge definiert werden, welche die Kompositionsschritte Identifikation, Bewertung und Komposition von Bausteinen ermöglichen. Der Bezug zum „Compositional Modeling“ wird im Zuge der Betrachtung dieser Themenbereiche in den entsprechenden Kapiteln dieser Arbeit im Detail durchgeführt.

Im Folgenden werden für die drei Modelle „Domain Theory“, „Scenario Description“ und „Query“ passende Gegenstücke aus dem Bereich der Benutzerschnittstellengenerierung gesucht, um einen ersten Einblick über die einzelnen Themenfelder, die in dieser Arbeit betrachtet werden müssen, zu ermöglichen.

Die „Scenario Description“ wird von Falkenhainer und Forbus (vgl. [FF91]) folgendermaßen definiert:

> “a scenario description: this includes its physical structure S and a (possibly empty) set of statements about its behaviour (e.g., initial conditions, steady-state, range limits, etc.);”

Eine „Scenario Description“ definiert demnach die Grundbausteine für die Generierung, indem es einerseits die beteiligten Objekte benennt, andererseits aber auch festlegt, wie diese Objekte miteinander verknüpft sind und wie diese sich verhalten. Aus einer „Scenario Description“ können dabei unterschiedliche Zielmodelle aufgebaut werden, die durch die Query bestimmt werden.

In dieser Arbeit wurden Interaktionsbeschreibungen zur Modellierung der „Scenario Description“ gewählt. Interaktionsbeschreibungen modellieren Interaktionen auf einem gewissen Abstraktionsniveau ohne beispielsweise Details über die Realisierung darstellen zu müssen. In Kapitel 5 wird diese Entscheidung motiviert, weiterhin werden notwendige Modelle für die Spezifikation der Interaktionsbeschreibung erstellt.

Die „Query“ wird von Falkenhainer und Forbus (vgl. [FF91]) folgendermaßen definiert:

> *" a query about the scenario's behaviour”*

Im Rahmen des „Compositional Modeling“ definiert die Query das eigentliche Ziel des Generierungsprozesses (in der Definition „Scenario Model“ genannt), nämlich auf welche Fragen das generierte Modell

[25] Strikte hierarchische Dekomposition des Modells; vollständige Beschreibung der Anforderungen durch eine Liste aus Modelltermen; für jedes Modell lässt sich eine Liste von Constraints ermitteln, die angeben, wann das Modell noch gültig ist.

Antworten liefern soll. Dieser Definition liegt dabei die Anwendungsdomäne der Thermodynamik zugrunde und muss auf die Benutzerschnittstellengenerierung übertragen werden.

Im Rahmen der automatisierten Generierung von Benutzerschnittstellen wird primär die Zielsetzung verfolgt, unterschiedliche Rahmenbedingungen zu berücksichtigen. Abhängig von den verfügbaren Endgeräten, von den aktuellen Aktivitäten, Präferenzen oder Behinderungen des Nutzers, aber auch von äußeren Einflüssen wie z.B. Lautstärke oder Lichtverhältnissen sollen effizient unterschiedliche der Situation angepasste Benutzerschnittstellen erzeugt werden.

Das Modell dieser Rahmenbedingungen bietet somit eine geeignete Analogie zur Query des „Compositional Modeling". In dieser Arbeit wird ein Kontextmodell (im Folgenden Nutzungskontextmodell genannt) herangezogen, um diese Rahmenbedingungen zu erfassen und schließlich soweit zu formalisieren, dass sie für eine automatische Generierung herangezogen werden können. In Kapitel 6 wird diese Entscheidung motiviert und notwendige Modelle für die Spezifikation der Kontextmodelle bereitgestellt.

Die „Domain Theory" wird von Falkenhainer und Forbus (vgl. [FF91]) folgendermaßen definiert:

> *"a Domain Theory Th, consisting of a set of domain model fragments* $\{m_1, \ldots, m_n\}$
> *and a set of rules constraining their use"*

Weiterhin sagen Falkenhainer und Forbus aus, dass eine „Domain Theory" eine Klasse von in Beziehung stehenden Phänomenen oder Systemen beschreibt. Modellfragmente („domain model fragments") sind dabei fundamentale Teile von Domäneneigenschaften wie Prozesse, Bauteile und Objekte und repräsentieren eine Wissensbasis, die durch den Kompositionsansatz zu neuem Wissen führen soll.

Die Übertragung des Begriffes „Domain Theory" in das Umfeld der Benutzerschnittstellengenerierung muss dabei die Fragestellung beantworten, wie eine Wissensbasis in diesem Bereich modelliert werden kann. Der Ansatz, der dabei in dieser Arbeit verfolgt wird, sieht vor, tatsächliche Realisierungen von Benutzerschnittstellen als Wissensbasis zu nutzen. Realisierung heißt dabei nicht, dass es sich um kompilierten Source-Code handeln muss, sondern dass eine Modellierungsmethodik gewählt werden muss, die direkt in ausführbare Benutzerschnittstellen überführt werden kann.

In Analogie zu dem in der Definition von Falkenhainer und Forbus verwendeten Begriff der „domain model fragments" werden in diesem Kapitel Benutzerschnittstellenfragmente spezifiziert, die wieder verwendbare Teile von Benutzerschnittstellen und ihre Möglichkeit zur Komposition beschreiben. Diese kapseln das Wissen bezüglich der Realisierung der Benutzerschnittstelle, beinhalten aber auch Wissen darüber, wie sich diese verhalten und welche Schnittstellen sie anbieten. In Kapitel 7 werden die notwendigen Modelle für die Spezifikation der Benutzerschnittstellenfragmente und schließlich der „Domain Theory" erarbeitet.

Ausgangslage für die Generierung der Benutzerschnittstelle ist die Aufgabenstellung nach Falkenhainer und Forbus. Basierend auf der „Scenario Description", der „Domain Theory" und der „Query" wird das

Zielmodell (das „Scenario Model“) generiert (vgl. [FF91]). Aus den eben vorgestellten Äquivalenten lässt sich die Aufgabenstellung wie folgt formulieren:

> *Gegeben seien eine Interaktionsbeschreibung in Form eines Abstract Interaction Nets (definiert in Abschnitt 5.3), eine „Domain Theory“ (definiert in Abschnitt 7.3) bestehend aus wieder verwendbaren Benutzerschnittstellenfragmenten (definiert in Abschnitt 7.2.1) und ein Nutzungskontext (definiert in Abschnitt 6.3).*
>
> *Aufgabenstellung ist es nun basierend auf den Vorgaben der Interaktionsbeschreibung durch Komposition der Elemente der „Domain Theory“ eine ausführbare Benutzerschnittstelle (definiert in Abschnitt 7.2.2.2) zu generieren, welche den durch den Nutzungskontext definierten Anforderungen bestmöglich genügt.*

In Kapitel 8 werden Methoden und Werkzeuge erarbeitet, mit dessen Hilfe diese Aufgabenstellung zu bewältigen ist.

4.2 Modelle und Werkzeuge für die Komposition

Basierend auf den drei Modellen „Interaktionsbeschreibung“, „Nutzungskontext“ und „Domain-Theory“ lassen sich bereits grob die für die Generierung notwendigen Bausteine benennen und darstellen, wie diese für den Generierungsprozess eingesetzt werden.

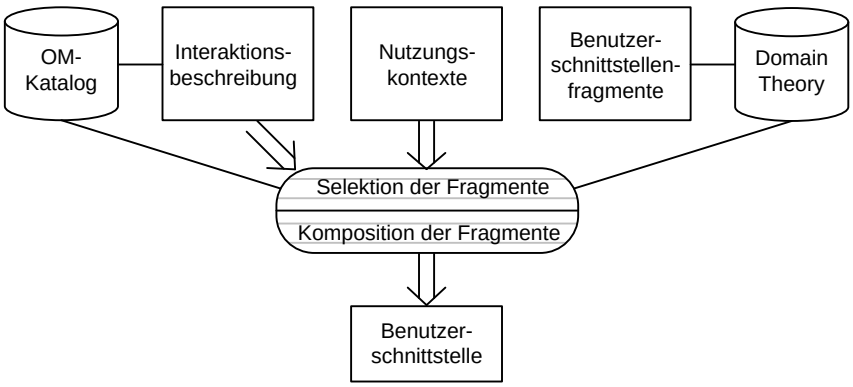


Abbildung 13: Modell des „Compositional Modeling“ für die Generierung von Benutzerschnittstellen

Abbildung 13 visualisiert den Blueprint des „Compositional Modeling“-Ansatzes für die Generierung von Benutzerschnittstellen. Die im Rahmen der Arbeit entwickelten Modelle (Interaktionsbeschreibung, Nutzungskontexte und Benutzerschnittstellenfragmente) sind als Rechtecke dargestellt, Bibliotheken (OM-Katalog und „Domain Theory“) als Zylinder und der Kompositionsprozess („Selektion der Fragmente“, „Komposition der Fragmente“) als Oval. Der Kompositionsprozess (in Abbildung 13 dargestellt durch das

Oval) unterteilt sich in Analogie zum „Compositional Modeling“-Ansatz in zwei Arbeitsschritte nämlich erstens die Selektion von auf den Nutzungskontext passenden Benutzerschnittstellenfragmenten der „Domain Theory“ und zweitens die Komposition der ausgewählten Benutzerschnittstellenfragmente, wie sie in der Interaktionsbeschreibung definiert wurde. Für jeden der beiden Schritte ist ebenfalls ein 3-stufiges Verfahren (Anforderungsanalyse, Spezifikation und Beispiel) vorgesehen.

Der Kompositionsprozess erhält für die Generierung als Input die Interaktionsbeschreibung und den Nutzungskontext (dargestellt durch die gerichteten Pfeile) und nutzt die beiden Bibliotheken („OM-Katalog“ und „Domain Theory“) als Wissensbasen (dargestellt durch ungerichtete Kanten). Der Entwicklungsprozess, der dem „Compositional Modeling“ zugrunde liegt, wird im folgenden Abschnitt 4.3 genauer beschrieben und dargestellt, wie Modelle verwendet und wie Bibliotheken für eine Anwendungsdomäne erstellt werden.

Der OM-Katalog (Kurzform für Objektmetaphern Katalog vgl. Kapitel 5.4) beschreibt eine Bibliothek, die als Werkzeug zur Erstellung der Interaktionsbeschreibung verwendet wird. Damit werden bereits auf Modellierungsebene „Bausteine“ bereitgestellt, die Wissen über die Anwendungsdomäne kapseln. Aufgebaut ist der OM-Katalog als katalogisierte Sammlung von Modellbausteinen, wie sie in der Interaktionsbeschreibung verwendet werden. Der Name „Objektmetaphern Katalog“ beschreibt den verwendeten Ansatz für die Modellierung der Interaktionsbeschreibung durch Metaphern.

Eine ausführbare Benutzerschnittstelle ist das Ergebnis der Generierung. Diese setzt sich aus Bausteinen der „Domain Theory“ zusammen und kann daher ebenfalls als ein Modell aufgefasst werden, das aber bereits soweit konkretisiert wurde, dass es im Rahmen einer Laufzeitumgebung direkt ausgeführt werden kann.

4.3 Entwicklungsprozess zum „Compositional Modeling“ von Benutzerschnittstellen

Ein „Compositional Modeling“-Ansatz zur Generierung von Benutzerschnittstellen kann nur effizient realisiert werden, wenn eine Spezialisierung auf eine Anwendungsdomäne getroffen wurde, welche die Komplexität des Baukastens beschränkt.

Im Rahmen eines Entwicklungsprozesses für die Generierung müssen folglich Vorarbeiten geleitet werden, in denen Werkzeuge bereitgestellt, Domänenwissen formalisiert und schließlich Anforderungen modelliert werden. Zu unterscheiden sind dabei vier Phasen im Entwicklungsprozess:

1. Aufbau eines „domänenspezifischen Baukastens für die Generierung von Benutzerschnittstellen“
   Dieser Arbeitschritt umfasst die Entwicklung und Konfiguration von Werkzeugen und Wissensbasen, die für die Generierung von Benutzerschnittstellen einer Anwendungsdomäne notwendig sind. Der Begriff der „Anwendungsdomäne“ ist für einen komponenten-basierten Ansatz wie das „Compositional Modeling“ entscheidend, da nur durch eine Fokussierung auf ein Aufgabenfeld die Komplexität soweit reduziert werden kann, dass die zur Verfügung stehenden Bausteine den Anforderungen ge-

recht werden. Der „domänenspezifische Baukasten“ bildet dabei eine Wissensbasis, die alle für die Benutzerschnittstellengenerierung einer Anwendungsdomäne relevanten Informationen kapselt. Der „domänenspezifische Baukasten“ ist dabei nicht als statische Wissensrepräsentation zu verstehen, sondern tatsächlich als ein Baukasten, der durch Nutzung (Modellierung der Anforderungen, Generierung der Benutzerschnittstelle) kontinuierlich erweitert und damit effizienter und für den Nutzer des Systems wertvoller wird.

2. Modellierung der Benutzerschnittstelle
   Die Generierung der Benutzerschnittstelle nach dem „Compositional Modeling“-Ansatz erfordert neben der „Domain Theory“, welche die eigentlichen Benutzerschnittstellenfragmente enthält, die Verfügbarkeit von Modellen, die das Verhalten und die Anforderungen an die Benutzerschnittstelle beschreiben. In diesem Entwicklungsschritt werden die benötigten Modelle mit Hilfe des „domänenspezifischen Baukastens“ erstellt. Um den Anspruch der Komposition und Wiederverwendung gerecht zu werden, ist der Modellierungsprozess eng mit der Nutzung des „domänenspezifischen Baukastens“ verbunden. So sind die in dieser Modellierung verwendeten Elemente immer auch Teil des „domänenspezifischen Baukastens“ und werden dort vollständig beschrieben.
3. Generierung der Benutzerschnittstelle
   Ausgangslage für die Generierung sind Nutzungsprofile, die Interaktionsbeschreibung und die „Domain Theory“. Der OM-Katalog wird als Wissensbasis und Bindeglied zwischen der Interaktionsbeschreibung und den Fragmenten der Domain Theory herangezogen. Der Generierungsprozess wird in Kapitel 8 eingehend betrachtet.
4. Evaluation und Konfiguration
   Das Ergebnis des Generierungsprozesses ist eine ausführbare Benutzerschnittstelle, welche in diesem Schritt bewertet und gegebenenfalls konfiguriert wird. Sollte sich das Ergebnis des Generierungsprozesses als soweit unzureichend erweisen, dass eine Konfiguration nicht mehr ausreichend ist, besteht die Möglichkeit, den Generierungsprozess erneut anzustoßen. Hierzu stehen dem Entwickler folgende Möglichkeiten zur Verfügung: Modifikation und/oder Erweiterung des „domänenspezifischen Baukastens“ um Benutzerschnittstellenfragmente, Modifikation der Nutzungskontexte, Modifikation der Interaktionsbeschreibung oder schließlich die Modifikation des OM-Kataloges.

In Abbildung 14 wird der kurz umrissene Entwicklungsprozess detaillierter dargestellt und insbesondere auf Aktivitäten, Werkzeuge und Modelle eingegangen, die im Entwicklungsprozess Verwendung finden. Zur Visualisierung des Entwicklungsprozesses wurde ein eEPK (erweiterte ereignisgesteuerte Prozesskette, vgl. [Sch01]) herangezogen. Um eine bessere Darstellbarkeit von Ereignissen, Funktionen, Modellen und Repositories zu gewährleisten, wurde weiterhin in Abbildung 14 folgende Farbkodierung gewählt: Ereignisse werden rot, Funktionen grün, Objekte des „domänenspezifischen Baukastens“ blau, Modelle für die Generierung gelb und schließlich die ausführbare Benutzerschnittstelle „gelblich grau“ dargestellt. Durch die geschweiften Klammern (links in Abbildung 14) wird die Zuordnung zu den vier oben beschriebenen Phasen im Entwicklungsprozess hergestellt.

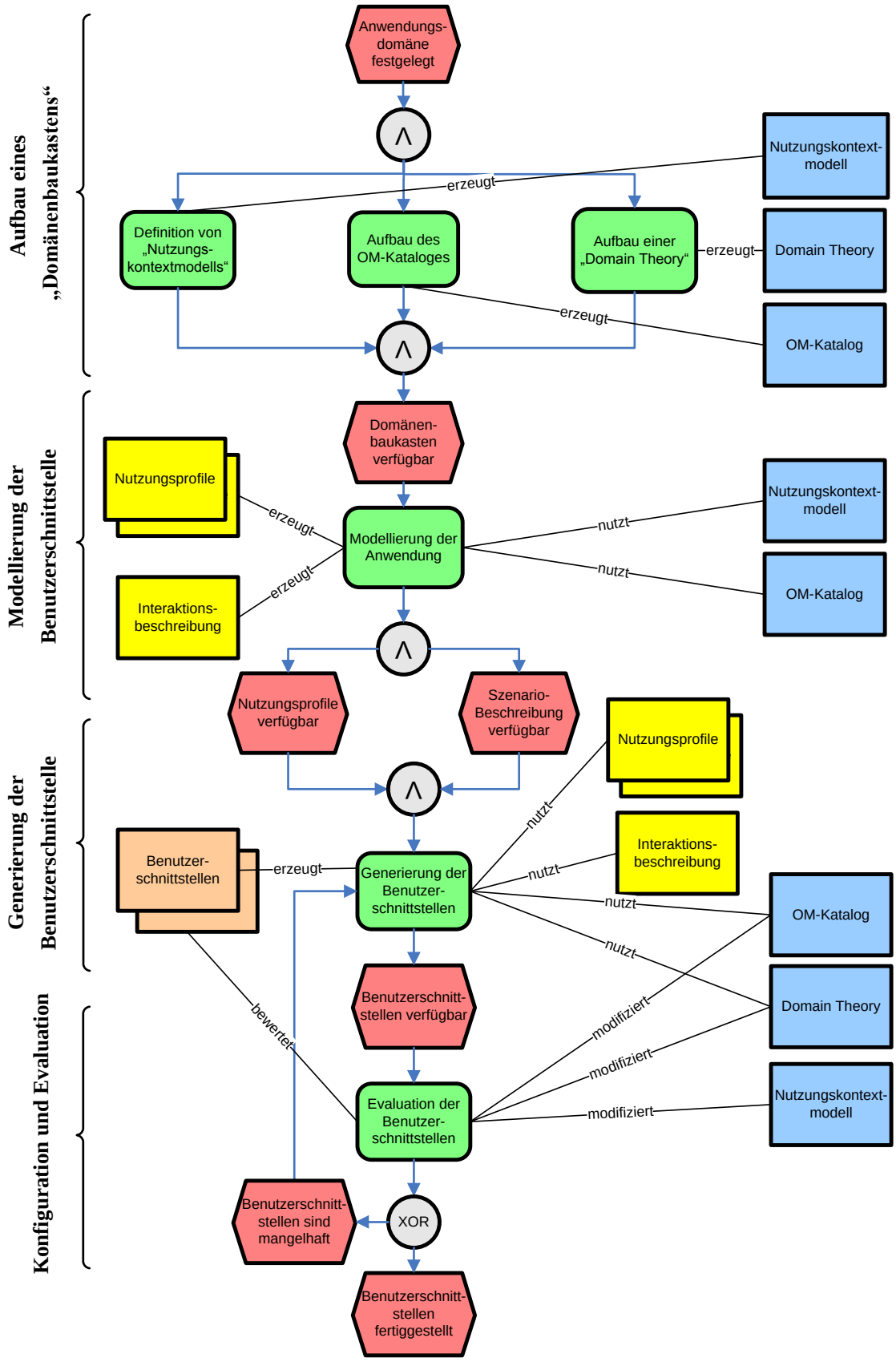


Abbildung 14: Entwicklungsprozess zum „Compositional Modeling“

Der „domänenspezifische Baukasten“ setzt sich aus einem Nutzungskontextmodell (eingeführt in Abschnitt 6.2.2), dem OM-Katalog (eingeführt in Abschnitt 5.4) und einer „Domain Theory“ (eingeführt in Abschnitt 7.3) zusammen.

Das Nutzungskontextmodell umfasst die Definitionen von Kontextelementen, die für die Beschreibung unterschiedlicher Nutzergruppen, Endgeräte und der Umgebungen der Nutzung für die betrachtete Anwendungsdomäne relevant sind. Weiterhin definiert das Nutzungskontextmodell die Relevanz von Kontextelementen für die Generierung der Benutzerschnittstelle. Mit Hilfe der Kontextelemente werden in der „Modellierungsphase“ Nutzungskontexte modelliert, für die Benutzerschnittstellen zu generieren sind. Diese Nutzungskontexte werden in die so genannten Nutzungsprofile überführt, die automatisiert verarbeitbar im Generierungsprozess eingesetzt werden.

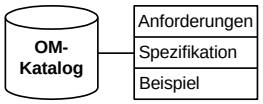
Dabei gilt, dass ein Nutzungsprofil für die Generierung genau einer Benutzerschnittstelle herangezogen wird. Dass in Abbildung 14 im Arbeitsschritt „Generierung der Benutzerschnittstelle“ eine Menge von Nutzungsprofilen angegeben ist, spiegelt eine wesentliche Restriktion des gewählten Ansatzes wieder. Es wird vorausgesetzt, dass im Vorfeld der Generierung bereits definiert wird, für welche Nutzungskontexte Benutzerschnittstellen generiert werden sollen. Dies bedeutet aber, dass im Vorfeld nicht nur festgelegt wird, welche Nutzer mit welchen Endgeräten und auch in welchen Umgebungen mit der Benutzerschnittstelle arbeiten werden, sondern auch welche Kombinationen dieser drei Komponenten auftreten werden. Ein Nutzungsprofil legt genau solch eine Kombination fest und definiert die Anforderungen, denen die Benutzerschnittstelle in diesem Fall gerecht werden muss. Das Ergebnis des Entwicklungsprozesses ist folglich eine Menge von Benutzerschnittstellen, wobei jeder Benutzerschnittstelle ein Nutzungsprofil zugeordnet ist. Dies bedeutet weiterhin, dass vor der Ausführung die Rahmenbedingungen (abgelegt im Nutzungsprofil) der Anwendung bekannt sein müssen.

Auch wenn ein flexibleres System erstrebenswert wäre, bei dem erst zur Laufzeit automatisiert Benutzerschnittstellen generiert werden, ist solch eine Lösung mit dem aktuellen Wissensstand nicht zu realisieren. Insbesondere die Notwendigkeit der Nachbearbeitung von Benutzerschnittstellen, für die der Arbeitsschritt „Evaluation und Konfiguration“ (vgl. Aktivität „Evaluation der Benutzerschnittstellen“ in Abbildung 14) eingeführt wurde, erfordert den Eingriff durch einen menschlichen Experten. An dieser Stelle ist noch erheblicher Forschungsaufwand zu leisten, um beispielsweise durch Erfahrungen im Praxiseinsatz und insbesondere der Verfeinerung der Wissensbasen (insbesondere „Domain Theory“ und OM-Katalog) auch die Evaluation automatisieren zu können. Im Rahmen der vorliegenden Arbeit wird dieser Bereich durch den Einsatz eines menschlichen Experten ausgeklammert.

Neben dem Nutzungskontextmodell sind der OM-Katalog und die „Domain Theory“ beide Bestandteile des „domänenspezifischen Baukastens“.

Der OM-Katalog wird in der „Modellierungsphase“ für die Erstellung der Interaktionsbeschreibung benutzt und ist auch in der Generierungsphase ein wichtiges Werkzeug.

Die „Domain Theory“ bildet schließlich die Wissensbasis, in der die Benutzerschnittstellenfragmente abgelegt sind und wird für die Generierung eingesetzt. In der Konfigurations- und Evaluationsphase kann durch Modifikation der „Domain Theory“ – beispielsweise durch die Modifikation oder die Erweiterung mit neuen Benutzerschnittstellenfragmenten – eine Grundlage für eine Neugenerierung geschaffen werden.



5 Interaktionsbeschreibung

In Kapitel 4 wurden im Rahmen der Entwicklung eines Blueprints Modelle und Verfahren definiert, die zur Verfügung zu stellen sind, um eine automatisierte Benutzerschnittstellengenerierung nach dem Leitbild des „Compositional Modeling“ zu ermöglichen. Abbildung 15 greift das in diesem Zusammenhang vorgestellte Modell der Bausteine des Generierungsprozesses (vgl. Abbildung 13) auf und visualisiert (blau hervorgehoben) die Teile, die innerhalb dieses Kapitels erarbeitet werden. Die Modelle sind weiterhin in drei Sektoren („Anforderungen“, „Spezifikation“ und „Beispiel“) unterteilt. Die Sektoren repräsentieren das Vorgehen bei der Herleitung und Beschreibung der Modelle, indem in einem ersten Schritt analysiert wird, welche Anforderungen an das Modell gestellt werden, anschließend eine Spezifikation der Modellierungsmethodik erfolgt und schließlich an einem Beispiel der Einsatz der Spezifikation veranschaulicht wird. Dabei wird als einheitliches Beispiel der in Abschnitt 3.3 eingeführte „Adipositas-Begleiter“ verwendet.

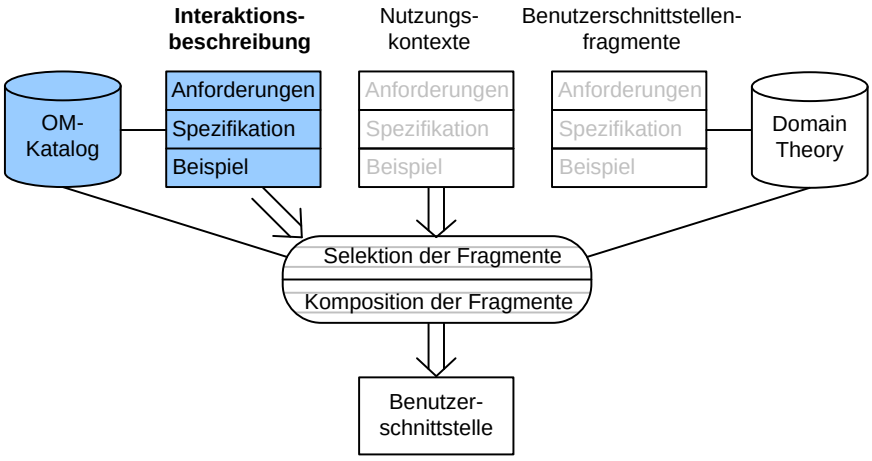
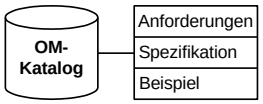


Abbildung 15: Einordnung der Interaktionsbeschreibung in den CM-Prozess

Die Interaktionsbeschreibung wird im Rahmen der Benutzerschnittstellengenerierung als Pendant zu der “scenario description” verwendet, die von Falkenhainer und Forbus (vgl. [FF91]) folgendermaßen beschrieben wurde:

> “a scenario description: this includes its physical structure S and a (possibly empty) set of statements about its behaviour (e.g. initial conditions, steady-state, range limits, etc.);”

Eine „scenario description“ definiert demnach die Grundbausteine für die Generierung, indem sie einerseits die beteiligten Objekte benennt, andererseits aber auch festlegt, wie diese Objekte miteinander verknüpft sind und wie sich diese verhalten.



Interaktionsmodelle eignen sich als Beschreibungsebene für die Benutzerschnittstellengenerierung, da sie Interaktionen auf einem geeigneten Abstraktionsniveau modellieren, ohne Details der Realisierung darstellen zu müssen. Hierzu wird in einem ersten Schritt der Interaktionsbegriff unterschiedlicher Modellierungsmethoden untersucht. Anschließend werden Anforderungen formuliert, die ein Interaktionsmodell für diese Arbeit erfüllen muss. Diese Anforderungen bilden die Grundlage für die Entwicklung einer eigenen Interaktionsmodellierung auf der Basis von „Metaphern“. Metaphern repräsentieren dabei Gedankenmodelle, die Anwender nutzen, um abstrakte Sachverhalte zu begreifen. Darauf aufbauend wird schließlich eine Modellierungsmethodik, die auf Petri-Netzen basiert, zur formalen Beschreibung eingeführt.

Basierend auf dieser Modellierungsmethodik kann schließlich die Konzeption und Einführung des OM-Kataloges (Kurzform für Objektmetaphern-Katalog) erfolgen. Dieser beschreibt eine Bibliothek, die als Werkzeug zur Erstellung der Interaktionsbeschreibung verwendet wird. Damit werden bereits auf Modellierungsebene „Bausteine“ bereitgestellt, die Wissen über die Anwendungsdomäne kapseln.

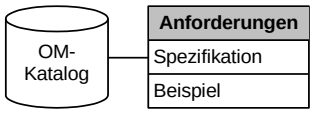
In einem letzten Schritt erfolgt schließlich die Evaluation der Modellierungsmethodik am Anwendungsbeispiel des „Adipositas-Begleiters“. Anhand eines konkreten Interaktionsbeispiels wird der Einsatz der Interaktionsbeschreibung unter Nutzung des OM-Kataloges vorgeführt.

5.1 Ziele und Anforderungen an eine Interaktionsbeschreibung

Die Interaktion von Menschen mit IT-Systemen lässt sich aus unterschiedlichen Perspektiven analysieren. Kognitive Modelle beispielsweise versuchen menschliche Denkmuster aufzugreifen, aber auch in der Soziologie und Psychologie ist der Interaktionsbegriff bekannt. Die folgenden Abschnitte bilden die Grundlage für eine Auseinandersetzung mit der Interaktionsmodellierung. In einem ersten Schritt wird hierzu der Interaktionsbegriff selbst betrachtet, um einen für die Zielsetzung der Arbeit passenden Interaktionsbegriff zu definieren. Basierend auf diesem Verständnis werden allgemeine Konzepte analysiert, welche die Mensch-Maschine-Interaktionen nicht mehr auf dem Level graphischer Benutzerschnittstellen abbilden, sondern in Form von Abstraktionen beschreiben. Aus dieser Betrachtung heraus werden Anforderungen abgeleitet, welche die Grundlage für eine kritische Betrachtung existierender Modellierungsansätze bilden.

5.1.1 Definition des Interaktionsbegriffs

Der Begriff der „Mensch-Maschine-Interaktion“ (englisch Man-Machine Interaction oder auch Human Computer Interaction) bezeichnet eine Disziplin in der Informatik, die sich mit der Evaluation und der Implementierung interaktiver Computersysteme für den Gebrauch durch Menschen und den damit ver-



bundenen Phänomenen beschäftigt (vgl. [ACM92]). Diese umfassende Definition fasst den Interaktionsbegriff sehr weit und bedarf einer Fokussierung auf die Problemstellung der vorliegenden Arbeit.

Interaktion ist laut Fremdwörterbuch (vgl. [DUDEN82]) ein in der Soziologie und der Psychologie geläufiger Terminus, mit dem das „aufeinander bezogene Handeln zweier oder mehrer Personen“ oder eine „Wechselwirkung zwischen Handlungspartnern“ bezeichnet wird. Basierend auf diesem Interaktionsbegriff beschreibt Norman in [Nor86] den Interaktionsprozess zwischen Mensch und Maschine in seinem „execution-evaluation cycle“ als Zyklus von sieben Stufen, der in zwei Phasen (Ausführung und Evaluation) unterteilt ist:

Ausführung:

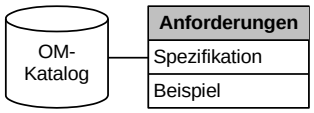
1. Festlegung eines Ziels
2. Formen der Intention
3. Spezifizieren einer Aktionssequenz
4. Ausführen der Aktionen

Evaluation:

1. Wahrnehmung des Systemzustandes
2. Interpretation des Systemzustandes
3. Evaluierung des Systemzustandes

Norman beschreibt dabei lediglich die Interaktion aus Sicht des Anwenders, trifft aber keine Aussagen über das System, mit dem der Anwender interagiert.

Eine Erweiterung dieses Modells hinsichtlich der Systemsicht repräsentiert das „interaction framework“ von Abowd und Beale’s in [AB91]. In diesem Framework sind der Benutzer und das System zwei Komponenten, die über Ein- und Ausgabekomponenten miteinander interagieren. Dabei durchlaufen sie, wie bei Norman, einen Zyklus aus Artikulation der Aufgabe, Verarbeitung, Präsentation und Beobachtung, aus der wiederum eine Artikulation von neuen Aufgaben folgt. Jeder Schritt dieses Zyklus verwendet dabei eigene Beschreibungen. So erfolgt die Artikulation der Aufgabe in einer Codierung, die dem Anwender geläufig ist, während die Verarbeitung des Systems sich einer gänzlich anderen Codierung bedienen wird. Als Schnittstelle zwischen System und Benutzer fungiert die Benutzerschnittstelle, welche die notwendigen Transformationen der unterschiedlichen Codierungen durchführen muss. Abbildung 16 veranschaulicht das „interaction framework“.



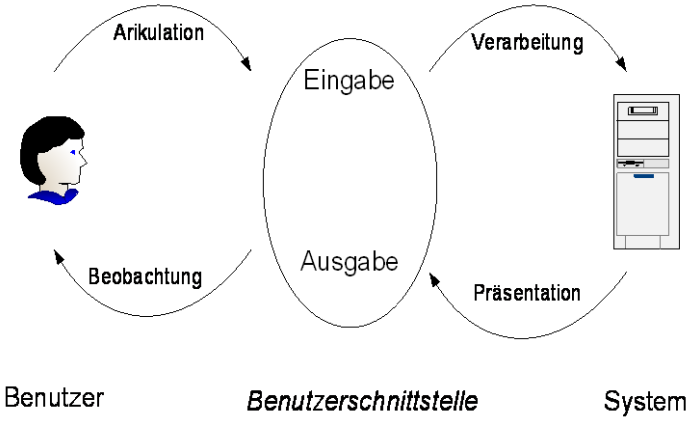
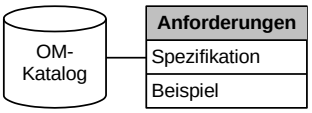


Abbildung 16: „Interaction Framework“ (in Anlehnung an [AB91])

Basierend auf diesem Verständnis eines „interaction framework“ wird der in dieser Arbeit verwendete Begriff der Benutzerschnittstelle und der Interaktion definiert. Unterschieden wird dabei zwischen der Benutzer- und Systeminteraktion. Die Benutzerinteraktion beschreibt dabei den linken Teil des „interaction frameworks“, welcher den Zyklus aus Artikulation der durchzuführenden Tätigkeit und der Beobachtung der Reaktion umfasst. Eine Benutzerinteraktion besteht dabei nicht notwendigerweise aus nur einem Zyklus aus Artikulation und Betrachtung, sondern umfasst vielmehr eine Vielzahl von Zyklen, die zur Durchführung einer bestimmten Aktivität dienen.

Die Systeminteraktion beschreibt den rechten Teil des „interaction frameworks“ in Abbildung 16, welcher den Zyklus aus der Verarbeitung der Nutzereingaben und die Generierung der Präsentation umfasst. Die Definition der Systeminteraktion ist analog zur Benutzerinteraktion insoweit zu verstehen, dass einer Nutzerinteraktion genau eine Systeminteraktion gegenübersteht, die aber aus mehreren Zyklendurchläufen bestehen kann. Die Systeminteraktion beschreibt daher die Aktionen des Systems, die für die Durchführung einer Aktivität aus Sicht des Anwenders benötigt werden.

Die Benutzerschnittstelle nimmt eine besondere Stelle in der Beschreibung von Interaktionen ein. Sie fungiert als Schnittstelle zwischen Benutzer- und Systeminteraktion und ist für die Synchronisierung der beiden Interaktionszyklen verantwortlich. Insbesondere wird durch die Benutzerschnittstelle definiert, welche Aktionen der Anwender durchführen kann und mit welchen Mitteln diese Aktionen ausgeführt werden. Die Präsentation als ein Schritt im Zyklus (siehe Abbildung 16) der Systeminteraktion modifiziert die Benutzerschnittstelle und verändert dadurch die Möglichkeiten des Benutzers zur Ein- und Ausgabe. Laut dieser Definition ist eine Benutzerschnittstelle ein dynamisches System, das basierend auf Benutzerinteraktionen und Systeminteraktionen agiert, und ist damit nicht auf physische Merkmale, wie die Interaktionsmedien (Maus, Tastatur, Spracheingabe, usw.) beschränkt.



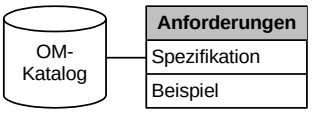
5.1.2 Existierende Interaktionsbeschreibungen

Die Modellierung von Interaktionen kann auf unterschiedlichen Ebenen erfolgen. So sind auch der im vorhergehenden Abschnitt vorgestellte „execution-evaluation cycle“ und das „interaction framework“ (vgl. Abbildung 16) Interaktionsmodelle, die mit dem Ziel erstellt wurden, mentale Prozesse zu erfassen, die in der Interaktion eine Rolle spielen. Ziel des „execution-evaluation cycle“ ist insbesondere die Identifikation von Problemstellungen im Umgang mit Anwendungen. Norman nutzt das Modell in [Nor86], um den so genannten „gulf of execution“ zu beschreiben. Damit versucht er Problemstellungen im Umgang mit Benutzerschnittstellen zu erfassen, die auftreten können, wenn die vom Nutzer geplanten Aktionen nicht mit den Möglichkeiten der Benutzerschnittstelle übereinstimmen.

In der Informatik ist die Modellierung von Benutzerschnittstellen ein häufig verwendeter Ansatz zur Interaktionsmodellierung. Modelliert werden dabei Elemente der Benutzerschnittstelle und damit auch die Interaktionsmöglichkeiten des Benutzers. Diese Modellierung kann dabei auf unterschiedlichen Abstraktionsebenen erfolgen. Visual-Tools (vgl. [Mye98]) bieten beispielsweise eine Umgebung, in der Benutzerschnittstellen graphisch modelliert werden. Das Modell kann direkt in ein ausführbares Programm umgewandelt werden, bietet aber einen sehr geringen Abstraktionsgrad.

Im Gegensatz dazu beschreiben Dialogmodelle (vgl.[Gre86], [BG04]) Interaktionen (mit einer Benutzerschnittstelle) mit Hilfe von formalen Spezifikationen, wie beispielsweise mit endlichen Automaten, Petri-Netzen und formalen Grammatiken. Dabei bedienen Sie sich unterschiedlicher Methoden als Abstraktion für eine Benutzerschnittstelle. Endliche Automaten (vgl. Abschnitt 5.2.4) bilden die Benutzerschnittstelle durch Zustände ab, die durch Interaktionen in einen anderen Zustand versetzt werden können. In Petri-Netzen (vgl. Abschnitt 5.2.4) repräsentieren beispielsweise die Stellen Fenster und die Transitionen Interaktionen in diesen Fenstern. Dialogspezifikationen werden zumeist in Zusammenhang mit einem so genannten UIMS (User Interfaces Management System) (vgl. [Mye98]) verwendet, das eine automatische Generierung eines Programms bzw. der Benutzerschnittstelle aus der Dialogspezifikation ermöglicht. Modellbasierte Ansätze für die Generierung von Benutzerschnittstellen (vgl. Abschnitt 2) setzen schließlich unterschiedliche Modelle für die Beschreibung einer Benutzerschnittstelle ein. Dabei ist das Dialogmodell nur eines dieser Modelle und wird durch Präsentations- (Modelle für die graphische Repräsentation), Aufgaben- (Modelle der Aufgaben, die mit der Benutzerschnittstelle bearbeitet werden sollen) und Endgerätemodelle (Modelle der Eigenschaften von Endgeräten) ergänzt.

Abstraktere Ansätze wie die Task-Modelle (vgl. [Pat01]) oder GOMS (vgl. [JK96]) verzichten nahezu völlig auf eine Beschreibung der Benutzerschnittstelle und modellieren Benutzerwünsche und Aufgaben, die zur Erfüllung dieser Wünsche durchzuführen sind. Diese Ansätze zeichnen sich durch eine Dekomposition von Aufgaben in Unteraufgaben aus, bis eine Granularität erreicht wird, die basierend auf einfachen Eingaben oder Selektionen auch für die Generierung von Benutzerschnittstellen verwendet werden kann.



In der Softwareentwicklung haben sich auch so genannte Use-Cases (vgl. [Oes05]) in der Anforderungsanalyse durchgesetzt. Use-Cases beschreiben Anwendungsfälle aus Sicht des Anwenders, insbesondere beschreiben sie, wie und mit welchem Ziel ein Anwender ein System nutzt. UML bietet mit Hilfe von Interaktionsdiagrammen und Interaktionsübersichtsdiagrammen Möglichkeiten zur Interaktionsmodellierung. Diese Diagramme modellieren nicht nur das Systemverhalten, sondern können auch für die Modellierung der Interaktion zwischen Mensch und Maschine eingesetzt werden. Das Ziel dieser Modelle ist primär die Spezifikation eines Anwendungssystems und die Formalisierung von Anforderungen.

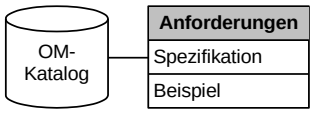
Die hier vorgestellten Ansätze beschreiben Interaktionen mit unterschiedlichen Zielsetzungen und auf sehr unterschiedlichen Abstraktionsebenen. Allen Ansätzen gemein ist, dass sie vollständig auf eine Betrachtung der physikalisch greifbaren Interaktionsmedien wie Maus und Tastatur verzichten, und diese implizit beispielsweise durch die Benennung von Interaktionsobjekten (Fenster, Buttons, Textfelder, Auswahllisten) einer graphischen Benutzerschnittstelle beschreiben.

Ein direkter Vergleich der Abstraktionsebenen der Interaktionsspezifikationen ist aber dennoch nicht möglich. So beschreiben Task-Modelle auf einem sehr hohen Abstraktionsniveau Ziele von Benutzern, die in Subziele zerlegt werden und schließlich in Aufgaben münden, die in einzelne Interaktionen zerlegt werden. Das weitere Vorgehen basiert auf einer Analyse der Anforderungen an eine Interaktionsmodellierung, wie sie in dieser Arbeit gefordert wird. Basierend auf diesen Anforderungen werden die unterschiedlichen Interaktionsansätze evaluiert.

5.1.3 Anforderungen an eine Interaktionsbeschreibung für die Benutzerschnittstellengenerierung

Eine Interaktionsmodellierung, wie sie in dieser Arbeit notwendig ist, verfolgt zwei konkurrierende Zielsetzungen: einerseits sollen möglichst detaillierte Informationen über und für die Realisierung der konkreten Benutzerschnittstelle modelliert werden, um ausführbare Benutzerschnittstellen aus der abstrakten Interaktionsmodellierung automatisiert generieren zu können; andererseits soll die Modellierung möglichst abstrakt gewählt werden, und eine konzeptionelle Beschreibung der Benutzerschnittstellen mit folgenden Zielsetzungen liefern:

- Reduktion der Komplexität;
- Analyse der Konzeption;
- Generierung unterschiedlicher Benutzerschnittstellen:
  - Benutzerschnittstellen für unterschiedliche Endgeräte (PC, PDA und Smart Phones);
  - Personalisierte Benutzerschnittstellen;
  - Kontextsensitive Benutzerschnittstellen;
- Wiederverwendung:
  - Portierung auf andere Anwendungsfälle;



- Identifikation von wieder verwertbaren Komponenten;

Dass es sich dabei um konkurrierende Zielsetzungen handelt, liegt auf der Hand. So beinhaltet eine realisierungsnahe Spezifikation der Benutzerschnittstelle viel mehr Informationen zur Umsetzung, ist aber auch erheblich komplexer, da Details der Realisierung mit in die Modellierung einfließen. Eine Analyse solch einer Spezifikation und schließlich die Portierung auf andere Benutzerschnittstellen ist höchst komplex, da gerätespezifische Eigenschaften meist untrennbar in der Spezifikation verankert sind. Neben der Portierung ist auch die Wiederverwendung von benutzerschnittstellennahen Spezifikationen problematisch, da wieder verwendbare Bausteine nur schwer zu identifizieren und zu extrahieren sind.

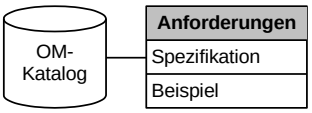
5.1.4 Analyse existierender Beschreibungsansätze

Basierend auf den in Abschnitt 5.1.3 eingeführten Zielsetzungen können einige Interaktionsansätze vernachlässigt werden, da sie für die grundsätzliche Zielsetzung, nämlich die Generierung von Benutzerschnittstellen, nicht konzipiert wurden. So fallen das „execution-evaluation cycle“ (vgl. [Nor86]), das „interaction framework“ (vgl. [AB91]) und auch die GOMS Modellierung (vgl. [JK96]) heraus, da sie Interaktionen konzeptionell betrachten und nicht die Zielsetzung verfolgen, Spezifikationen zu liefern, die direkt ausführbar sind. Gleiches gilt für die Modellierungsansätze aus UML2, die ebenfalls die Interaktionsmodellierung zur Anforderungsanalyse verwenden und auch keine automatische Generierung von Benutzerschnittstellen vorsehen.

Die bekanntesten Ansätze, die dieser Forderung entsprechen, sind einerseits UIMS (vgl. Abschnitt 5.1.4.1), die historisch eine der ersten Ansätze zur abstrakten Beschreibung von Interaktionen mit der Zielsetzung der automatischen Benutzerschnittstellengenerierung liefern. Weiterhin sind Task-basierte Ansätze (vgl. Abschnitt 5.1.4.2) zu erwähnen, die sowohl über eine Methodik zur Problemanalyse, als auch über Möglichkeiten zur Dialogspezifikation verfügen. Modellbasierte Ansätze (vgl. Abschnitt 5.1.4.3) nutzen schließlich unterschiedliche Modelle zur Spezifikation. Auf eine Betrachtung von deklarativen Benutzerschnittstellenspezifikationen, die heutzutage XML als Beschreibungssprache einsetzen, wird verzichtet, da sich hinter den Beschreibungssprachen Task- und insbesondere modellbasierte Ansätze verbergen.

5.1.4.1 Interaktionsmodellierung mittels UIMS

User Interface Management Systeme (kurz UIMS) sind einer der ersten Ansätze für eine formale Spezifikation von Benutzerschnittstellen, aus der ausführbare Anwendungen generiert werden können. Der Begriff UIMS ist dabei in Analogie zu DBMS (Database Management System) entstanden, und soll die Idee eines DBMS für die Verwaltung und Organisation von Datenbeständen auf Benutzerschnittstellen übertragen.



Das Seeheim Modell (vgl. [Ol92]) bildet die Basis eines UIMS. Das Seeheim Modell teilt die Benutzerschnittstelle in drei funktionale Komponenten auf: Präsentationskomponente, Dialogkontrolle und die Applikationsinterfacekomponente. Die Präsentationskomponente bildet die Schnittstelle zum Benutzer und ist für Ein- und Ausgaben des Anwenders verantwortlich. Die Dialogkontrolle steuert die gesamte Anwendung, indem sie Daten aus der Applikationskomponente an die Präsentationskomponente übergibt und andererseits Daten aus der Präsentationskomponente an die Applikationskomponente weiterleitet. Die gesamte Steuerung des Systems obliegt dabei der Dialogkontrolle, die sowohl die Übergaben steuert, als auch entscheidet, wie die Benutzerschnittstelle auf Interaktionen des Anwenders reagiert.

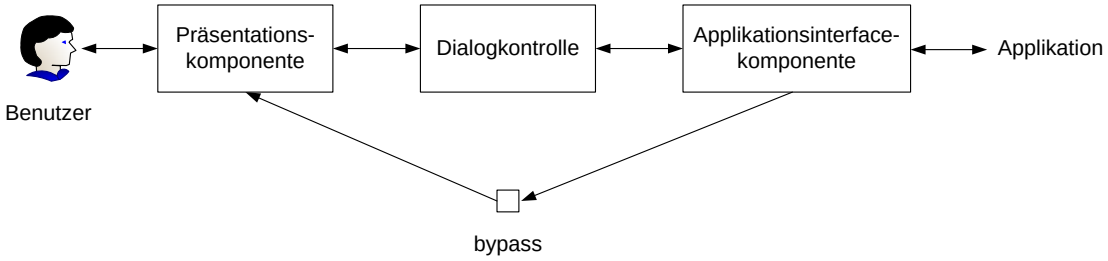


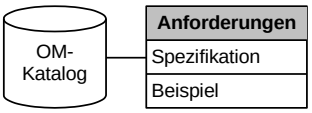
Abbildung 17: Seeheim-Modell (in Anlehnung an [Ol92])

Abbildung 17 veranschaulicht die Kommunikation der einzelnen Komponenten und stellt die Rolle der Dialogkontrolle als zentrale Steuerungsinstanz der Benutzerinteraktion dar. Eine Erweiterung des Modells wurde hierbei über den „bypass“ eingeführt. Dieser Mechanismus ermöglicht es, direkt und damit ohne den Umweg über die Dialogkontrolle Daten in die Präsentation zu integrieren. Diese Erweiterung wird für interaktive Systeme benutzt, wenn Änderungen im Datenbestand direkt auf der Benutzerschnittstelle visualisiert werden sollen.

Ziel eines UIMS ist die Spezifikation der Benutzerschnittstelle mittels einer Dialogspezifikation, welche Sequenzen von Ereignissen und notwendige Interaktionsformen beschreibt. Die in den UIMS verwendete Dialogspezifikation ermöglicht somit die Modellierung von Interaktionen auf einem höheren Abstraktionslevel (als bei der Programmierung). Für die Entwicklung von User Interfaces sind keine Programmierkenntnisse notwendig, sondern lediglich Kenntnisse über die verwendete Dialogspezifikation. Es existieren eine Vielzahl von UIMS, die sich im Wesentlichen in Bezug auf ihre Dialogspezifikation unterscheiden.

Dialogspezifikationen nutzen unterschiedliche Modellierungsmethodiken wie Zustandsübergangsdiagramme, Petri-Netze, Graphtransformationen, Grammatiken, aber auch eventbasierte Beschreibungssprachen und Constraints werden für die Modellierung eingesetzt.

Am Beispiel von Zustandsübergangsdiagrammen als typische Dialogspezifikationssprache soll die Semantik einer Dialogspezifikation vorgestellt werden.



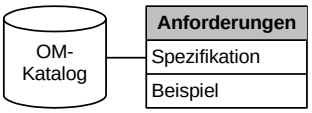
Ein Zustandsübergangsdiagramm ist eine graphische Darstellung eines endlichen Zustandsautomaten. Dialogspezifikationen durch Zustandsübergangsdiagramme beschreiben die Dialogzustände, den zulässigen Dialogablauf und die Verarbeitung von Benutzereingaben durch Knoten, Kanten und Markierungen in gerichteten Graphen. Die Knoten des Graphen repräsentieren Dialogzustände, die Kanten die Zustandsübergänge. Kanten beschreiben die zulässigen Interaktionsmöglichkeiten in den einzelnen Dialogzuständen und werden durch die Namen der Ereignisse (Benutzereingaben) markiert, die einen entsprechenden Zustandsübergang auslösen. Weiterhin können den Kanten Aktionen zugewiesen werden, die bei dem entsprechenden Zustandsübergang ausgeführt werden.

Erweiterungen des Konzeptes nach Green in [Gre86] sind:

- Modulare Zustandsübergangsdiagramme, die eine Zerlegung des Graphen in mehrere Untergraphen vorsehen.
- Rekursive Zustandsübergangsdiagramme, die modulare Zustandsübergangsdiagramme um die Möglichkeit der rekursiven Aktivierung von Untergraphen erweitern. Dabei können Kanten eines Untergraphen mit dem Namen des Untergraphen markiert werden, so dass durch Rekursion mehrere aktive Instanzen dieses Untergraphen erzeugt werden können.
- Erweiterte Zustandsübergangsdiagramme, deren Kanten neben Ereignissen auch Bedingungen zur Überprüfung der Zulässigkeit einer Transition zugeordnet werden. Hierdurch ist es möglich das dynamische Verhalten des Systems anhand von Registerwerten zu beschreiben, die sich auf den Dialogablauf auswirken.

Zustandsübergangsdiagramme eignen sich primär für die Beschreibung von zustandsorientierten Eingabetechniken wie Menüs, Eingabemasken (Formulare) und Kommandosprachen. Problematisch ist jedoch die Modellierung direktmanipulativer graphischer Benutzerschnittstellen (Tabellenkalkulation, Bildbearbeitung, CAD-Tools). Diese besitzen aufgrund der Vielzahl von Interaktionsmöglichkeiten eine große Anzahl an Zuständen und Zustandsübergängen, die sich nicht mehr in einem Zustandsübergangsgraphen abbilden lassen. Weitere Mängel sind das Fehlen globaler Interaktionsmöglichkeiten wie Hilfefunktionen oder Abbruchinteraktionen und das Fehlen von Konzepten zur Modellierung von Nebenläufigkeit, wie sie für die Beschreibung von zwei oder mehr aktiven Teildialogen benötigt werden. Ein Zustandsübergangsdiagramm müsste hierzu jeden möglichen Zustand des Systems erfassen und abbilden, was zu einer nicht mehr beherrschbaren Menge an Zuständen führen würde.

Aufgrund dieser Problemstellungen nutzen neuere Dialogspezifikationssprachen Petri-Netze (vgl. [WL08]) oder Statecharts, die zwar mit einer sehr ähnlichen Semantik versehen sind, aber insbesondere die Problemstellung der Parallelität berücksichtigen.



Zusammenfassend ist festzuhalten, dass Dialogspezifikationen den notwendigen Detaillierungsgrad bereitstellen, um automatisiert Benutzerschnittstellen generieren zu können. Modelliert wird dabei das Verhalten einer realen Benutzerschnittstelle. Zwar erfolgt die Modellierung nicht direkt anhand von Elementen der Benutzerschnittstelle, dennoch bildet beispielsweise ein Systemzustand in einem Zustandsüber-Zustandsübergangsdiagramm eine logische Einheit und damit eine Präsentation der Benutzerschnittstelle ab.

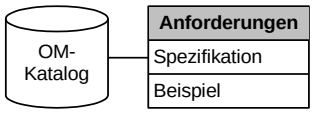
5.1.4.2 Task-Modelling

Task-Modelle basieren auf der hierarchischen Dekomposition wie sie GOMS (Goals, Operators, Methods, Selection Rules) (vgl. in [CMN84]) zugrunde liegt. GOMS modellieren die Interaktion des Benutzers mit einem Computersystem, indem in einem ersten Schritt die Ziele (Goals) des Benutzers identifiziert werden. Jedes dieser Ziele wird anschließend in „Sub-Ziele“ zerlegt, die zur Erreichung dieses Ziels notwendig sind. Die Methoden (Methods) beschreiben Verfahren, die zur Zielerreichung durchzuführen sind. Diese setzen sich wiederum aus elementaren Operatoren und schließlich Regeln für die Auswahl aus einer Menge von Operatoren zusammen. GOMS Modelle wurden primär für die Analyse und Performance-Messung der Interaktionen eines Benutzers im Umgang mit einem ihm bekannten Informationssystem entwickelt (vgl. [CMN84], [OO91]).

Task-Modelle konzentrieren sich auf die hierarchische Dekomposition von Tasks zur Erreichung eines bestimmten Ziels. Ziele können logische Aktivitäten sein, wie die Suche nach einer bestimmten Information, aber auch physische Aktivitäten, wie das Bestellen eines Taxis (vgl. [Pat01]). Dabei wird Task definiert als “an activity that should be performed in order to reach a goal” (vgl. [MPS02]). Task-Modelle beschreiben die unterschiedlichen Aktivitäten, die in der Interaktion mit einem Informationssystem durchzuführen sind (vgl. [ST04]). Tasks und Ziele sind dabei eng verwoben. Für die Zielerreichung können dabei ein oder mehrere Tasks durchzuführen sein, gegebenenfalls können auch alternative Tasks existieren. Ein Task kann wiederum in Sub-Tasks zerlegt werden, das sind wieder Teilaufgaben, die zur Durchführung des übergeordneten Tasks ausgeführt werden müssen.

Für die Modellierung von Task-Modellen existieren unterschiedliche Notationen. Exemplarisch wird das CTT (Concurrent Task Tree) (vgl. [Pat99]) vorgestellt, welches über entsprechende Erweiterungen verfügt, die es erlauben, aus dem Task-Modell Benutzerschnittstellen zu generieren. Die CTT-Notation wird auch häufig in modellbasierten Ansätzen für die Modellierung des Task-Modells verwendet, so nutzt Theresa (vgl. [BCP04]) CTTE (Concurrent Task Tree Editor) für die Modellierung von Task-Modellen. CTT erweitert einfache hierarchische Task-Modelle einerseits durch die Definition von Task-Kategorien und andererseits durch eine Kontrollstruktur auf einer Hierarchieebene (vgl. [Pri96]). Folgende Task-Kategorien werden dabei unterschieden [PMM97]:

- User Task
  Tasks, die ausschließlich vom Benutzer ohne Unterstützung eines Informationssystems durchgeführt werden.



- Abstract Task
  Ein komplexer Task, der unterschiedliche Tasks beinhaltet oder nicht mit den anderen Task-Kategorien beschreibbar ist.
- Interaction Task
  Ein Task, der durch den Benutzer mit Hilfe eines Informationssystems durchgeführt wird.
- Application Task
  Ein Task, der ausschließlich von einem Informationssystem durchgeführt wird.

Weiterhin steht eine Reihe von Kontrollstrukturen zur Verfügung. Die folgende Tabelle führt die Strukturen mit einer kurzen Beschreibung ein.

- Überlappung (T1 ||| T2)
  Tasks können unabhängig von ihrer Reihenfolge abgearbeitet werden.
- Synchronisation (T1 |[]| T2)
  Tasks haben in ihrem Ablauf eine Synchronisationsstelle und hängen voneinander ab.
- Reihenfolge (T1 >> T2)
  Erst wenn ein Task abgeschlossen ist, kann der andere starten.
- Synchronisation mit Informationsübergabe (T1 []>> T2)
  Zusätzlich zu der Synchronisation werden Informationen vom ersten an den folgenden Task übergeben.
- Alternative (T1 [> T2)
  Wenn der erste Task durchgeführt wird, kann der zweite nicht mehr durchgeführt werden.
- Iteration (T1*)
  Der Task wird mehrfach durchgeführt.
- Optionaler Task ([T])
  Dieser Task kann, muss aber nicht durchgeführt werden.
- Rekursion T
  Ermöglicht den rekursiven Aufruf von Tasks.

Anzumerken ist, dass diese Kontrollstrukturen immer nur sequenziell auf einer Hierarchieebene angewendet werden. Abbildung 18 visualisiert ein CTT, das einen Ausschnitt aus dem Adipositas-Begleiter (vgl. Abschnitt 3.3) modelliert.

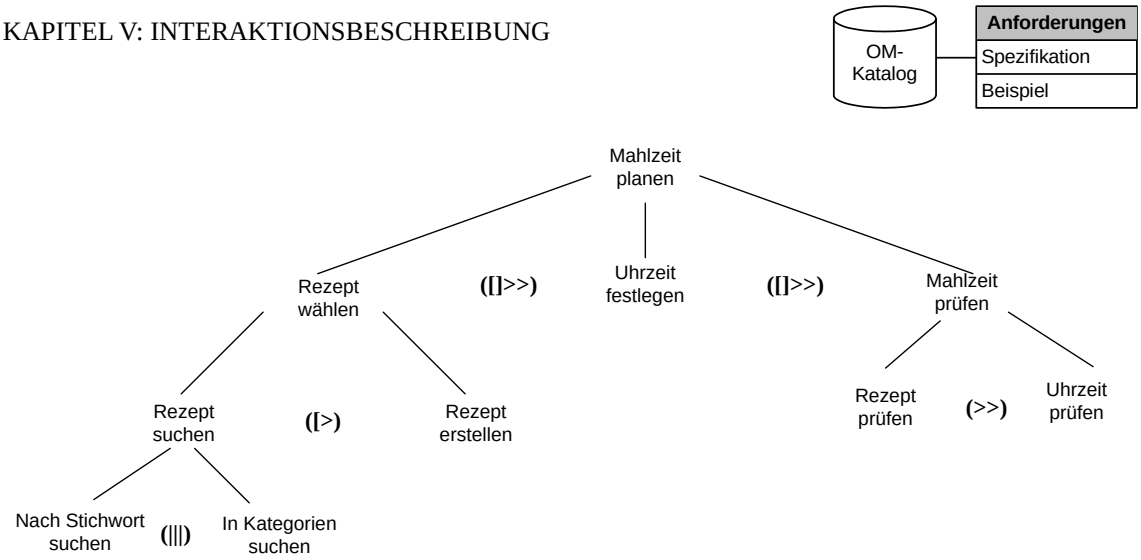


Abbildung 18: Concurrent Task Tree am Beispiel des Adipositas-Begleiters

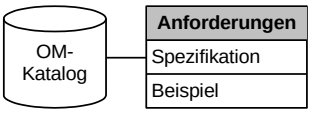
Der Task „Mahlzeit erstellen“ wird hier in drei Sub-Tasks („Rezept wählen“, „Uhrzeit festlegen“ und „Mahlzeit prüfen“) zerlegt. Auf dieser Hierarchieebene lassen sich folgende Relationen definieren: so müssen der Reihe nach ein Rezept ausgewählt werden, eine Uhrzeit angegeben werden und anschließend die Mahlzeit (bestehend aus Rezept und Uhrzeit) geprüft werden. Dabei liefert jeweils der Vorgänger Inhalte für den nachfolgenden Task. Im Gegensatz dazu kann nur einer der Tasks „Rezept suchen“ oder „Rezept erstellen“ (beides Sub-Tasks von „Rezept auswählen“) durchgeführt werden.

CTTs ermöglichen eine abstrakte Modellierung von Interaktionen auf Basis von Tasks und den Relationen zwischen den Tasks einer Hierarchieebene. Die Beschreibung ist dabei unabhängig von der Benutzerschnittstelle, da die definierten Aufgaben sich ausschließlich an Tätigkeiten, nicht aber an Benutzerschnittstellenelementen oder Schritten orientieren. Zusätzlich bieten die unterschiedlichen Task-Kategorien eine Möglichkeit auch Tasks des Systems zu modellieren. So würde das „Nach Stichworten suchen" (in Abbildung 18) ein Application Task sein, der durch die Angabe eines Suchbegriffes initiiert werden würde.

Der große Vorteil der Task-Modellierung, nämlich der hohe Abstraktionsgrad und die Unabhängigkeit von konkreten Benutzerschnittstellen, ist aber auch der wesentliche Nachteil für die Generierung der Benutzerschnittstelle. So können in Task-Modellen lediglich Informationen über Folgen und Strukturen der Dateneingabe bzw. der Selektion von Alternativen modelliert werden.

5.1.4.3 Modellbasierte Ansätze

Eine Analyse von Beschreibungsansätzen für die Interaktionsmodellierung muss modellbasierte Systeme für die Benutzerschnittstellengenerierung mit einschließen. Für eine eingehende Betrachtung sei auf Kapitel 2 dieser Arbeit verwiesen, in welchem solche Systeme vorgestellt und untersucht wurden.



5.1.4.4 Zusammenfassung und Resümee

Die untersuchten Ansätze weisen Stärken aber auch Schwächen in gewissen Bereichen auf. So haben User Interface Management Systeme mächtige Werkzeuge zur abstrakten Spezifikation von Benutzerschnittstellen und liefern auch direkt die Möglichkeit, Benutzerschnittstellen auf Basis dieser Spezifikationen zu erstellen. Sie haben aber Schwächen aufgrund der sehr schnittstellennahen Spezifikation. Während die Stärken der Dialogspezifikationssprachen auf der Ebene der Benutzerschnittstellenmodellierung liegen, haben Task-Modelle ihre Stärke in der Strukturierung von komplexen Problemfeldern und bieten durch die Definition von Zielen einen starken Einbezug von Nutzerwünschen in die Modelle. Andererseits bietet die Dekomposition bis herunter auf elementare Interaktionen wie Eingaben von Daten und Auswählen von Alternativen nur eine sehr schwache Beschreibung der Benutzerschnittstelle. Auch die Erweiterungen des Ansatzes um Kontrollstrukturen für Abläufe und Alternativen können nichts an der Problemstellung der mangelnden Nähe zur Benutzerschnittstelle ändern. Abbildung 19 visualisiert diese Problemstellung in einem Graphen. Auf der X-Achse wird der Abstraktionsgrad der Beschreibung abgebildet, dabei ist links eine höchst abstrakte Beschreibung vorgesehen, die nach rechts konkreter wird bis hin zur tatsächlichen Realisierung. Auf der Y-Achse wird abgebildet, wie gut ein Werkzeug die Beschreibung auf einer bestimmten Abstraktionsebene unterstützt. Dabei gilt: je höher der Wert ist, desto besser gestaltet sich die Unterstützung. Die beiden Kurven in Abbildung 19 visualisieren die Unterstützung der Abstraktionsgrade durch CCT und UIMS. Der Task-basierte Ansatz am Beispiel von CTT bietet eine sehr gute Unterstützung in der abstrakten Beschreibung von Aufgaben und deren Strukturierung. Die Unterstützung wird aber schwächer, je genauer spezifiziert werden soll, was auf Ebene der Benutzerschnittstelle geschieht. Die Abbildung gibt wieder, dass es durchaus möglich ist, Benutzerschnittstellen aus CTTs zu erzeugen, hierzu aber weitere Informationen über den Dialog hinzugezogen werden müssen. UIMS haben ihre Stärken in der Generierung ausführbarer Benutzerschnittstellen und der Modellierung mit Hilfe von Dialogspezifikationen. Dabei ist der Abstraktionsgrad von der Dialogspezifikation abhängig. Höhere Abstraktionen sind aber mit den bisherigen Dialogspezifikationen nicht darstellbar.

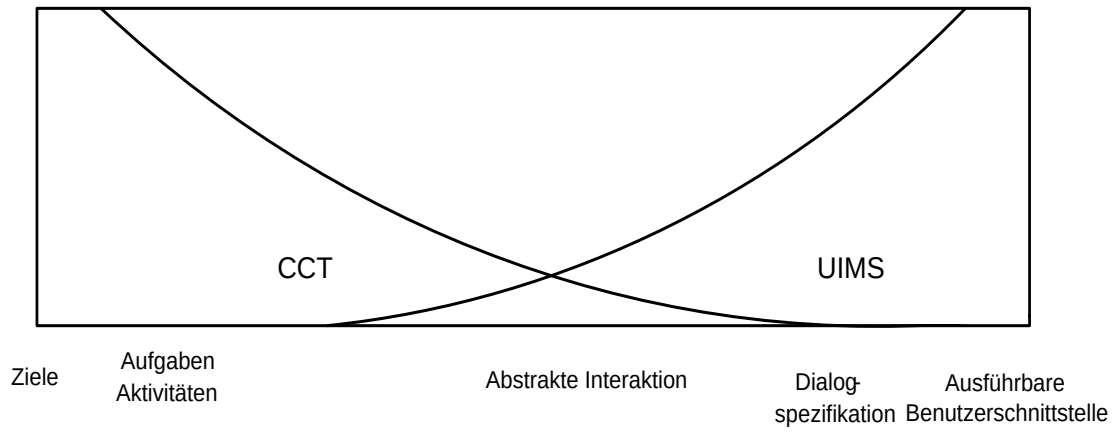
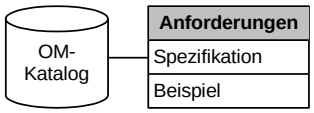
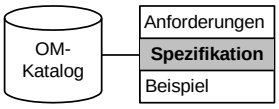


Abbildung 19: Gegenüberstellung von CTT und UIMS zur abstrakten Beschreibung von Benutzerschnittstellen



Modellbasierte Ansätze versuchen diesen Graben zwischen der Spezifikation der Benutzerschnittstelle und der Modellierung von abstrakten Abläufen zu überbrücken, indem beide und zusätzliche Modelle für die Beschreibung herangezogen werden. Durch eine geschickte Verknüpfung dieser Modelle wird versucht, der Problemstellung zu begegnen. So werden einzelnen Tasks im Task-Modell Dialogspezifikationen zugeordnet, die dann tatsächlich eine Benutzerschnittstellen-Beschreibung enthalten. Ein automatisiertes Matching ist schwierig und nur unter Ausnutzung bestimmter Restriktionen überhaupt möglich, wie in [LVN00] gezeigt wird.

Versucht man die Position des Grabens auf Basis der Abstraktionsskala (Zieldefinition bis zur ausführbaren Benutzerschnittstelle) zu identifizieren, so stellt man fest, dass Modelle für die Beschreibung von „abstrakten Benutzerschnittstellen“ fehlen. Diese „abstrakten Benutzerschnittstellen“ sollen auf der Ebene der Benutzerschnittstellen modellieren, aber unabhängig von konkreten Realisierungen sein. Der in Abschnitt 5.2 vorgestellte Ansatz versucht diesen Graben auszufüllen, und damit eine abstraktere Modellierung bereitzustellen, aus der ausführbare Benutzerschnittstellen generiert werden können.



5.2 Interaktionsmodellierung mit Objektmetaphern

Die Modellierung von abstrakten Interaktionen muss unabhängig von der Benutzerschnittstelle geschehen, aber dennoch Interaktionskonzepte liefern, die sich in einer konkreten Realisierung niederschlagen. Der in dieser Arbeit eingeschlagene Weg orientiert sich an der Gedankenwelt des Benutzers, der mit dem System interagiert. Es wird die Vision verfolgt, Vorstellungen, die Menschen mit bestimmten Diensten verknüpfen, aufzugreifen und diese als Basis für die Interaktionsmodellierung zu verwenden. Die Gedankenwelt des Anwenders sollte sich optimalerweise direkt in der Benutzerschnittstelle widerspiegeln, um ihm den Umgang mit der Anwendung zu erleichtern.

Als Ansatz zur Realisierung dieser Vision wurde die Modellierung von abstrakten Interaktionen mit Hilfe von Metaphern herangezogen. Metaphern repräsentieren dabei Gedankenmodelle, die Benutzer verwenden, um abstrakte Sachverhalte (also auch Dienste eines Informationssystems) zu begreifen. Das Begreifen ist dabei wörtlich zu interpretieren, indem sich Metaphern zumeist auf real existierende Umgebungen, Objekte, Beziehungen und Gesetzmäßigkeiten beziehen.

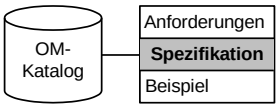
In den folgenden Kapiteln wird ein Metamodell für die Interaktionsmodellierung mit Hilfe von Metaphern erarbeitet. Hierzu wird in einem ersten Schritt der Metaphernbegriff, wie er in der Informatik verwendet wird, untersucht und anschließend basierend auf diesen Betrachtungen eine Methodik konzipiert, die beschreibt, wie Metaphern für die Interaktionsmodellierung genutzt werden können. In den darauf folgenden Abschnitten werden Vorarbeiten für die Auswahl einer geeigneten Spezifikationsmethodik geleistet. Basierend auf den Anforderungen, die sich aus der Interaktionsmodellierung mit Metaphern ableiten lassen, werden existierende Modellierungsansätze der Interaktionsmodellierung als formale Interaktionsspezifikation untersucht und bewertet.

5.2.1 Definition von Metaphern und deren Bedeutung in der Informatik

Im allgemeinen Sprachgebrauch wird eine Metapher als eine rhetorische Figur verstanden, bei der das Wort nicht in seiner eigentlichen, sondern übertragenen Bedeutung gebraucht wird (vgl. [DUDEN82]). Klassische Beispiele für diese Form von Metapher sind Ausdrücke wie „Rabeneltern“, „jemanden in den Himmel loben“ oder „auf den Hund kommen“. Oftmals wird auch die Bildhaftigkeit von Metaphern hervorgehoben.

Eine weiterreichende Bedeutung der Metapher[26] stellten Lakoff und Johnson erstmals in [LJ80] vor. Metaphern sind demnach nicht rein sprachliche sondern insbesondere auch kognitive Phänomene und somit auf der Ebene der Kognition (kognitive Metapherntheorie) angesiedelt. Nach dieser Definition sind sie

[26] Lakoff und Johnson [LJ80] nutzen den Begriff der „konzeptuellen Metapher“ (conceptual metaphor), um sich von der Metapher als rhetorische Figur abzugrenzen.

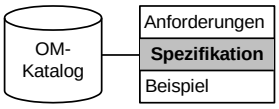


ein wesentliches Mittel, um abstrakte bzw. unbekannte Sachverhalte erfassen und begreifen zu können. Dabei werden konkrete Erfahrungen eines Menschen auf abstrakte Sachverhalte übertragen. Metaphern stellen somit konzeptuelle Verbindungen zwischen Sinn- und Erfahrungsbereichen her. In [LJ80] wird dabei zwischen zwei „Bereichen“ unterschieden: einerseits der Herkunftsbereich (source domain), der „das für einen Menschen Bekannte“ abbildet und andererseits der Zielbereich (target domain), welcher eine unbekannte bzw. abstrakte Umgebung beschreibt, die es zu begreifen gilt. Metaphern sind somit ein kognitives Mittel zum Verständnis von Entitäten und Zusammenhängen, die auf der Übertragung von bereits existierendem Wissen basiert. Eine eingehende Analyse des Metaphernbegriffs und der damit verbundenen kognitiven Prozesse findet sich in [Fen03].

Die Bedeutung dieses Metaphernbegriffs für das Verständnis von Computersystemen, die eine höchst komplexe und gleichzeitig auch abstrakte Domäne darstellen, wird in [Hän04] thematisiert. Hier wird anschaulich am Beispiel von Computer-Fachzeitschriften dokumentiert, wie stark der verwendete Wortschatz durch den Einsatz von Metaphern geprägt ist.

Auch für die Beschreibung von Computer Programmen (Anwendungen) ist der Einsatz von Metaphern unverzichtbar. Eine der populärsten Metaphern ist die so genannte „Desktop-Metapher“ (beispielsweise beim Microsoft Windows Betriebssystem). Als „source domain“ wurde der klassische Büroarbeitsplatz mit seinen Objekten, wie Ordner, Mülleimer, Schreibtisch, Dokumente, Verweise, usw. gewählt. Die Verwendung der Metaphern geht dabei über die Beschreibung der Objekte hinaus. So werden auch Aktivitäten mit Hilfe von Metaphern greifbar gemacht. Ein Nutzer kann beispielsweise Dokumente löschen, indem er diese in den Papierkorb legt. Den Papierkorb kann er schließlich leeren und damit die Daten endgültig zerstören. Mit dieser Metapher wird der relativ komplexe Vorgang des zeitlich verzögerten Löschens von Daten (bzw. deren Wiederherstellung) mit einem im Büroalltag gängigen und begreifbaren Vorgang verknüpft, und damit ein Verständnis der Vorgänge des Computer-Systems ermöglicht.

Metaphern werden dabei nicht nur für die Benutzer von Computer-Systemen verwendet, sondern auch für Anwendungs-Designer und Programmierer. So wird die Metaphernbildung in der objektorientierten Softwareentwicklung als eines der wesentlichen konzeptionellen Werkzeuge angesehen. Objekte der Wirklichkeit werden herangezogen, um Probleme zu modellieren und sie in einer Programmiersprache abzubilden. Objektorientierte Softwaresysteme können hierbei als eine Menge von Objekten in Form von Klassen und deren Beziehungen verstanden werden. Dieses Konzept wird insbesondere deshalb als besonders vielversprechend und intuitiv angesehen, da hier eine natürliche und wirklichkeitsnahe Modellierung einer Problemstellung realisiert wird. In der Objektorientierung wird zwischen unterschiedlichen Entwicklungsstufen unterschieden, in denen von einer abstrakten Beschreibung (OO-Analyse) bis hin zur Implementierung (OO-Programmierung) die objektorientierte Sichtweise beibehalten wird.



Die Objektorientierung in der Gestaltung der Benutzerschnittstelle ist dabei besonders hervorzuheben. Hier sind die einzelnen Elemente einer Benutzerschnittstelle (Fenster, Dialog, Buttons, Textfelder, Menüs usw.) Objekte in einer Programmiersprache, die ein bereits definiertes Verhalten vorweisen und vom Entwickler wieder verwendet werden können. Für den Benutzer einer Benutzerschnittstelle besitzen Metaphern eine weiter reichende Bedeutung. Die Benutzerschnittstelle stellt den einzigen Zugang des Nutzers zur Anwendung dar, über die er alle Interaktionen durchführen kann. Erst durch die Verwendung von Metaphern erhalten diese Oberflächenelemente tatsächlich für den Nutzer eine Bedeutung und ermöglichen es, Erwartungen zum Umgang bzw. zur Interaktion mit diesen Elementen zu verbinden.

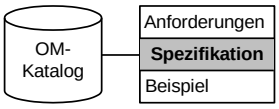
Auch in neuen Ansätzen im Bereich der Softwareentwicklung, wie beispielsweise „Extreme Programming“, werden Metaphern bewusst eingesetzt ([KBN*04]). Eine Metapher ist hier eine von den zwölf so genannten „core practices“. Diese beschreiben grundlegende Praktiken, wie „Refactoring“, „testgetriebene Entwicklung“, „einfaches Design“ usw. Metaphern werden im Extreme Programming explizit als Mittel zur Kommunikation mit Außenstehenden, insbesondere den Kunden, verwendet. Metaphern (auch Systemmetaphern genannt) stehen für die grundsätzliche Idee hinter der Architektur und dienen zur Kommunikation sowohl für Entwickler als auch für Kunden. Im Extreme Programming ersetzen Systemmetaphern komplexe Software-Architekturen. Die Entwickler sollen sich von der Metapher leiten lassen. Dabei sollen die verwendeten Metaphern über den gesamten Entwicklungsprozess beibehalten werden, und eine einheitliche Sprache für Kunden und Entwickler bieten.

Einen besonderen Stellenwert nehmen Metaphern in neuartigen Benutzerschnittstellen ein, wie beispielsweise Benutzerschnittstellen, die auf der Manipulation in dreidimensionalen Räumen beruhen. Im Gegensatz zu den bekannten zweidimensionalen Benutzerschnittstellen, die bereits mit bekannten „Interaktionsobjekten“ (Textfeld, Icon, Menü) besetzt sind, hat man in einem dreidimensionalen Raum die Möglichkeit, völlig neue Objekte und Interaktionen mit diesen Objekten zu definieren. Hier spielen wiederum Metaphern eine wesentliche Rolle, denn diese neuartigen Interaktionen müssen vom Benutzer zu allererst verstanden werden.

Das folgende Kapitel beschreibt einen Ansatz, wie Benutzerschnittstellen mit Hilfe von Metaphern beschrieben werden können.

5.2.2 Beschreibung von Interaktionen mit Hilfe von Metaphern

Der Einsatz von Metaphern in der Softwareentwicklung und insbesondere für die Konzeption und Entwicklung von Benutzerschnittstellen ist heutzutage zu einem festen Bestandteil geworden. Metaphern für Benutzerschnittstellen werden dabei primär eingesetzt, um dem Benutzer einen intuitiven Zugang zu der Benutzerschnittstelle und den Interaktionsmöglichkeiten zu geben. Metaphern haben aber auch das Potential, als Basis für eine abstrakte Spezifikation von Interaktionen zu dienen. Sie bieten die Möglichkeit, abstrakte Tätigkeiten zu beschreiben, die ein Nutzer über eine Benutzerschnittstelle durchführen kann, indem die Semantik aus einer bekannten Domäne entliehen wird. Eine Übertragung der Metapherndefini-



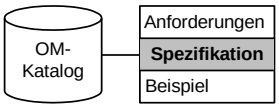
tion nach Lakoff und Johnson in [LJ80] auf die Interaktionsmodellierung liegt dabei auf der Hand und kann auf Basis der Grundbegriffe „source domain“ und „target domain“ durchgeführt werden.

Die abstrakte Umgebung (target domain) ist die Benutzerschnittstelle, die als einziger Zugangspunkt des Nutzers zur Anwendung fungiert. Die „source domain“ ist schließlich das dem Benutzer Bekannte, aus dem er Sachverhalte für und über die Benutzerschnittstelle ableiten kann. Die Modellierung von Interaktionen mittels Metaphern bezieht sich somit primär auf die Modellierung der Interaktionen in der „source domain“. Der Begriff der „source Domain“ ist dabei schwer zu formalisieren. Lakoff beschreibt die „source domain“ als etwas begreifbares, wobei begreifbar im wörtlichen Sinne verwendet wird, also etwas, dass man tatsächlich anfassen und modifizieren kann. Um nicht ein vollständiges Modell einer Domäne für die Spezifikationen von Interaktionen aufbauen zu müssen, muss definiert werden, was das Objekt der Modellierung ist.

So wird nur ein sehr eingeschränkter Teil der eigentlichen „target domain“ für die Modellierung von Interaktionen benötigt. In einer „source domain“, die als etwas physikalisch greifbares beschrieben wird, liegt es nahe, Interaktionen mit realen Objekten in Beziehung zu setzen, die der Nutzer manipulieren oder zu einem gewissen Zweck benutzen kann. Diese Manipulation bzw. Nutzung beschreibt schließlich die Interaktionen, die ein Benutzer durchführt. Die Möglichkeiten des Nutzers zur Interaktion werden dabei durch die ihm zur Verfügung stehenden Objekte eingeschränkt. Durch Interaktionen können nun neue Interaktionsobjekte verfügbar gemacht oder existierende deaktiviert werden. Die Semantik der Objekte und der angebotenen Interaktionen auf diesen Objekten ist durch die „source domain“ definiert, und sollte den Erfahrungen des Nutzers in dieser Domäne entsprechen. Daher ist die Wahl der „richtigen“ Domäne für die Beschreibung einer Anwendung entscheidend.

Bei Anwendungen, die aus der Perspektive eines Anwenders Dienste sind und ihm eine gewisse Leistung anbieten, ist die Wahl der „source domain“ zumeist trivial, da oftmals eine direkte Beziehung zu einem „realen“ Sachverhalt herangezogen werden kann.

Da sich einerseits im Bereich der Telemedizin Metaphern noch nicht etabliert haben und andererseits auch vermittelt werden soll, dass Metaphern bereits heute erfolgreich eingesetzt werden, wurde ein Beispiel aus der Anwendungsdomäne der „Internet-Shops“ entliehen. Als „source domain“ wird hier von vielen Anbietern ein Kaufhaus herangezogen. Nahezu alle großen Anbieter von Internet Shops wie Quelle, Otto, Neckermann, Amazon usw. bedienen sich dabei ähnlicher Metaphern. Tabelle 1 listet typische Objektmetaphern auf, die in diesen Anwendungen Verwendung finden.



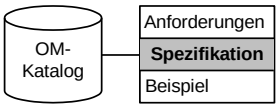
Source Domain	Objektmetaphern	Interaktionen	Erzeugt/Entfernt Objekt
Kaufhaus	Warenkorb	Warenkorb bearbeiten	
		Warenkorb einsehen	
		Zur Kasse	Kasse
	Katalog	Produkt suchen	Produktliste
		Kategorie auswählen	Katalogkategorie
	Produkt	Betrachten	
		Modifizieren	
		Zum Warenkorb hinzufügen	
	Produktliste	Produkt auswählen	Produkt
	Katalogkategorie	Kategorie auswählen	
	Rechnung	Betrachten	
	Kasse	Rechnung erstellen	Rechnung
		Wahl der Lieferung	
		Authentifizierung	Kunde
		Registrieren	Kunde
		Wahl der Zahlungsmethode	Konto/Kreditkarte/ Nachnahme
	Konto	Konto festlegen	
		Rechnung bezahlen	
	Kreditkarte	Kreditkarte festlegen	
		Rechnung bezahlen	
	Kunde	Authentifizieren	Kunde
		Zugangsdaten festlegen/ändern	
		Basisdaten festlegen/ändern	

Tabelle 1: Beispiele für Objektmetaphern in der Source Domäne Kaufhaus

Neben den allgemeinen Metaphern finden sich in den unterschiedlichen Internet Shops noch eine Vielzahl weiterer Objektmetaphern, die spezielle Angebote des Shops repräsentieren. Beispiele für solche Angebote sind die „Expresskasse“ oder der „Direkt-Kauf (1-Klick©)“, aber auch diverse Angebotsseiten. Auch hier wird deutlich, dass Metaphern verwendet werden, um dem Benutzer einen intuitiveren Zugang zu den Funktionalitäten des Systems zu gewähren.

5.2.3 Grundlagen für die Interaktionsmodellierung mit Objektmetaphern

Für die Modellierung von Interaktionen mit Hilfe von Metaphern wird eine Methodik benötigt, die es erlaubt, wesentliche Aspekte der Interaktion zu erfassen und in einem Modell abbilden zu können. Ein erstes Beispiel wurde in Abschnitt 5.2.2 präsentiert. In diesem Kapitel wird ein Metamodell vorgestellt, das die wesentlichen Aspekte des Modellierungsansatzes beschreibt, und schließlich Ausgangsbasis für



eine konkrete Modellierungsmethodik sein wird. Dabei werden grundsätzliche Modellierungskonzepte und -elemente vorgestellt und diskutiert.

5.2.3.1 Metamodell der Modellierung mit Objektmetaphern

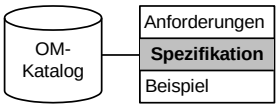
Der Begriff des Metamodells wird in der Literatur unterschiedlich definiert. Allgemein wird ein Modell, das ein anderes beschreibt, Metamodell genannt. Für ein Metamodell, das die Interaktionsmodellierung mit Hilfe von Metaphern beschreibt, wird die Definition von Fettke und Loos in [FL03] herangezogen. Diese verweisen in Ihrer Definition auf [Str96], nach dem Metamodelle zwei Prinzipien verfolgen können:

- Ein sprachbasiertes Metamodell repräsentiert die bei der Konstruktion des zu beschreibenden Modells verwendete Modellierungssprache.
- Ein prozessbasiertes Metamodell repräsentiert die bei der Konstruktion des zu beschreibenden Modells durchgeführten Modellierungsschritte.

In dieser Arbeit werden ausschließlich sprachbasierte Metamodelle mit dem Begriff Metamodell assoziiert. Sprachbasierte Metamodelle sind Modelle, welche die Konstrukte und deren Beziehung beschreiben. Im Gegensatz zu Fettke und Loos, die ihr Metamodell als Formalismus zur Beschreibung der Modellierung verwenden, dient das hier definierte Metamodell lediglich dem Verständnis der Modellierungselemente und ihrer Beziehungen zueinander.

Abbildung 20 visualisiert nach dieser Definition ein Metamodell für die Interaktionsmodellierung mit Metaphern. Rechtecke repräsentieren die Konstrukte der Modellierung und die Kanten die unterschiedlichen Beziehungen. Für die Beschreibung werden zwei unterschiedliche Typen von Beziehungen verwendet:

- Zuordnung
  Beschreibt eine Klassifikationsbeziehung. So ist jede Objektmetapher und jeder Interaktionsraum genau einer „source domain“ zugeordnet.
- Aggregation
  Beschreibt die Zusammensetzung eines Objektes aus anderen. So ist ein Interaktionsraum durch eine Kollektion aus Objektmetaphern definiert.



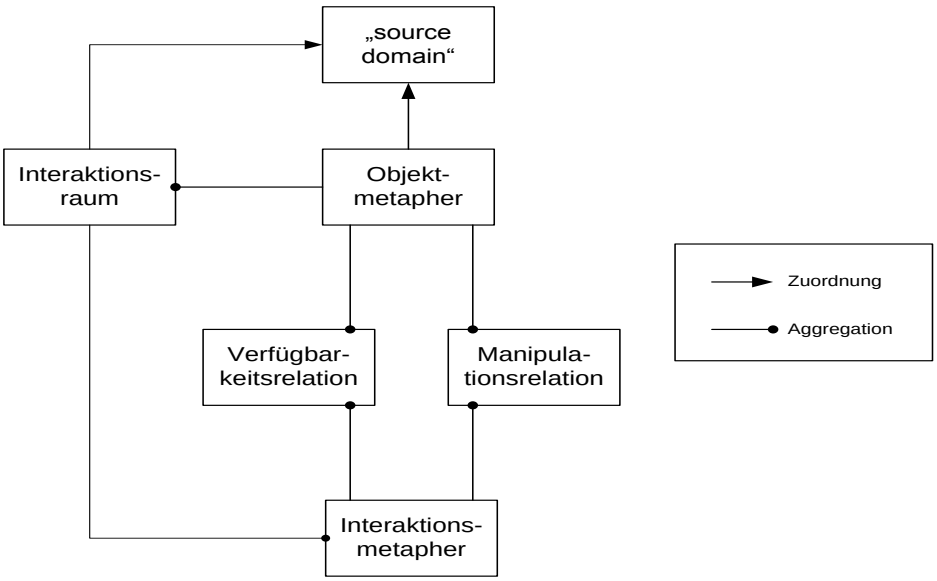


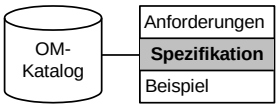
Abbildung 20: Meta-Modell für die Interaktionsmodellierung

5.2.3.2 Source Domain

Die „source domain“ orientiert sich an der Definition des hier verwendeten Metaphernbegriffs nach Lakoff [LJ80]. Somit ist eine „source domain“ eine dem Benutzer bekannte, zumeist auch tatsächlich materiell greifbare Umgebung. Eine klare Definition, welche Aspekte und Eigenschaften eine „source domain“ enthalten muss, ist in der Literatur nicht zu finden. Lakoff nutzt für die Beschreibung der „source domain“ den Begriff der „conceptual domain“. Damit beschreibt er die „source domain“ als Menge von zusammengehörigen Konzepten, die Objekte, aber auch Regeln, Beziehungen und Verhalten sein können.

In der Interaktionsmodellierung soll eine „source domain“ abgrenzbare Bereiche definieren, die aus Objekt- und Interaktionsmetaphern bestehen. Die „source domain“ definiert somit eine Art Leitmetapher, aus der schließlich Objekte und Interaktionen für die Modellierung abgeleitet werden können. Zieht man das in Tabelle 1 präsentierte Beispiel für die Verwendung von Metaphern heran, so wäre das Kaufhaus die entsprechende „source domain“. Die richtige Wahl dieser Leitmetapher kann dabei durchaus problematisch sein. Madsen [Mad94] beschreibt die Problemstellungen einer systematischen Herangehensweise für eine Metaphernwahl und stellt erste Richtlinien für die Entwicklung von Metaphern bereit. Er bezeichnet dabei seinen Ansatz insoweit als pragmatisch, dass er bewusst auf Formalismen zur Beschreibung von Metaphern verzichtet.

Auch für diese Arbeit wird ein pragmatischer Ansatz gewählt, indem die „source domain“ als ein Katalog von Objekt- und Interaktionsmetaphern beschrieben wird, die in einem definierten Bereich in einer Beziehung zueinander stehen. Da die „target domain“ ein Informationssystem ist, das den Anwender in einer bestimmten Lebenslage bzw. bei der Bewältigung einer Aufgabe aus dem realen Leben unterstützen soll, ist der erste Ansatz für die Wahl der „source domain“ immer das entsprechende Problemfeld im realen Leben.



Sowohl der Interaktionsraum, als auch die dort verwendeten Objektmetaphern gehören zur selben „source domain“. Damit ist sichergestellt, dass die Modellierung (der Interaktionsraum) und auch die verwendeten Modellierungselemente derselben Domäne entstammen.

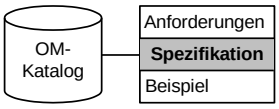
5.2.3.3 Interaktionsraum

Der Interaktionsraum definiert für den Modellierer eine Umgebung, die sich an der realen Welt orientiert und innerhalb der er eine entsprechende Anwendung beschreiben möchte. Zieht man an dieser Stelle wieder das in Tabelle 1 präsentierte Beispiel für die Verwendung von Metaphern heran, so wäre beispielsweise der „Amazon Internet-Shop“ der Interaktionsraum, den ein Modellierer gestalten kann, und sich dabei aus den Objektmetaphern der „source domain“ Kaufhaus bedienen kann. Im Gegensatz zur „source domain“, die Objekt- und Interaktionsmetaphern einer Domäne beschreibt, ist der Interaktionsraum eine Instanz der Objekte der „source domain“, in der Interaktionen und ihre Resultate modelliert werden. Der Interaktionsraum definiert dabei insbesondere, welche Interaktionen für den Benutzer in einem bestimmten Zustand des Interaktionsraums möglich sind. Modelliert werden diese Möglichkeiten über die Verfügbarkeit von Objektmetaphern in einem Interaktionsraum. Interaktionen können nun diesen Interaktionsraum verändern, indem weitere Objektmetaphern verfügbar gemacht werden, oder aber aus dem Interaktionsraum entfernt werden. Der Interaktionsraum bedient sich somit selbst einer „Raummetapher“ für die Beschreibung der Modellierung. Zugrunde liegt der Interaktionsraum, der mit Objekten gefüllt werden kann. Welche Interaktionen nun ein Benutzer durchführen kann, ist von den Objekten in diesem Raum abhängig. Dabei verändert sich der Raum, wenn Interaktionen durchgeführt werden. Diese Veränderungen können die bereits existierenden Objekte betreffen, aber auch neue Objekte können durch Interaktion in diesem Raum erzeugt werden. Anzumerken ist, dass nur das Zerstören oder Erzeugen von neuen Objektmetaphern die Interaktionsmöglichkeiten beeinflusst, während die Modifikation von Objektmetaphern zwar modelliert wird, aber keine direkte Auswirkung auf die Interaktionsmöglichkeiten des Benutzers hat.

5.2.3.4 Objekt- und Interaktionsmetaphern

Objekt- und Interaktionsmetaphern beschreiben die Elemente des Interaktionsraums. Jede Objektmetapher ist einer „source domain“ zugeordnet und wird durch die Interaktionsmöglichkeiten definiert. Die Beziehung zu Interaktionen wird dabei durch zwei Relationen (Verfügbarkeits- und Manipulationsrelation) definiert.

Die Verfügbarkeitsrelation modelliert die Notwendigkeit von Objektmetaphern für die Durchführung einer Interaktion. Dabei können auch mehrere Objektmetaphern für die Durchführung einer Interaktion benötigt werden. So kann beispielsweise die Interaktion „essen“ nur dann verfügbar sein, wenn die Objektmetaphern „Messer“, „Gabel“, „Teller“ und auch die „Mahlzeit“ selbst verfügbar sind. Welche Objektmetaphern tatsächlich verfügbar sind, wird durch den Interaktionsraum definiert. Die Verfügbarkeitsrelation modelliert lediglich die Regel der Notwendigkeit für die Durchführung von Interaktionen.



Die Manipulationsrelation modelliert schließlich die Auswirkungen einer Interaktion auf andere Objektmetaphern. Auswirkungen können dabei die Modifikation existierender Objektmetaphern, aber auch die Erzeugung von neuen oder die Zerstörung bereits existierender Objektmetaphern in einem Interaktionsraum sein. Modelliert werden diese Auswirkungen über eine Beziehung zu Objektmetaphern, die an der Durchführung der Interaktionen betroffen sind. Dabei können die Mengen der beiden Relationen (Verfügbarkeits- und Modifikationsrelation) überschneidend sein. Dieses kann aber auch heißen, dass Interaktionen Objektmetaphern für ihre Durchführung benötigen und diese bei der Durchführung zerstören oder modifizieren können.

Die Aggregationsrelation zwischen Interaktionsraum und Interaktionsmetapher modelliert die Möglichkeit, komplexen Interaktionen einen eigenen Interaktionsraum zuzuordnen. Diese Zuordnung erlaubt es dem Modellierer, die Durchführung von Interaktionen genauer zu spezifizieren, indem ein eigener Interaktionsraum zur Beschreibung der Interaktionsmetapher verwendet wird. Der so definierte Interaktionsraum besteht initial aus den Objektmetaphern, die der Interaktionsmetapher über die Verfügbarkeitsrelation zugeordnet sind. Jeder dieser Objektmetaphern können beliebige Interaktionsmetaphern zugeordnet werden, über die der „Unter-Interaktionsraum“ verändert werden kann. Dabei gelten dieselben Regeln, wie sie für Interaktionsräume definiert wurden. Ein „Unter-Interaktionsraum“ ist abgeschlossen, sobald alle Objektmetaphern verfügbar gemacht und/oder zerstört wurden, so wie es in der Interaktionsmetapher (die den „Unter-Interaktionsraum“ definiert) spezifiziert wurde.

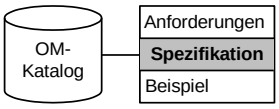
Diese „Unter-Interaktionsräume“ ermöglichen auch tiefergehende hierarchische Strukturierungen, indem auch Interaktionen eines „Unter-Interaktionsraum“ wiederum einem Interaktionsraum zugeordnet werden. Es bleibt dabei dem Modellierer überlassen, welche der Interaktionen er als komplex ansieht und durch einen Interaktionsraum weiter spezifiziert.

5.2.4 Modellspezifikationen für die Interaktionsmodellierung

Die in Abschnitt 5.2.1 anhand eines Metamodells vorgestellte Interaktionsmodellierung unterscheidet sich in einigen Aspekten erheblich von existierenden Interaktionsmodellen und Benutzerschnittstellen-Spezifikationen. Die Unterschiede werden anhand von Beispielen existierender Modellierungsspezifikationen diskutiert.

5.2.4.1 UML

Betrachtet man UML in der Version 2.0 (im Folgenden UML2 (vgl. [Oes05]) genannt), die einen großen Fundus an Modellierungstechniken bietet, so lassen sich die Diagrammtypen in Struktur- und Verhaltensdiagramme unterteilen. Strukturdiagramme können dabei bereits im Vorfeld aus dieser Betrachtung ausgeschlossen werden, da sie ausschließlich statische Aspekte eines Softwaresystems beschreiben. Verhaltensdiagramme hingegen modellieren das dynamische Verhalten und eignen sich daher prinzipiell für die Modellierung von Interaktionen. UML2 (im Vergleich zu der Vorgängerversion) bringt wesentliche Erweiterungen und Verbesserungen in den Verhaltensdiagrammen, und damit für die Modellierung des



dynamischen Verhaltens. Betrachtet man die sieben Verhaltensdiagramme (Anwendungsfall-, Aktivitäts-, Sequenz-, Kommunikations-, Interaktionsübersichts-, Zeitverlaufs- und Zustandsdiagramme), so beschreiben sechs davon Abläufe im Software-System. Eine Sonderrolle nehmen die Anwendungsfalldiagramme[27] ein, da sie Anwendungsfälle und die beteiligten Akteure modellieren. Dabei hat der Modellierer die Möglichkeit, Anwendungsfälle weitergehender zu beschreiben. Primär werden Anwendungsfalldiagramme für die Spezifikation von Anforderungen an das System verwendet. Für die Modellierung des eigentlichen Verhaltens werden Verhaltensmodelle eingesetzt.

Die Modellierung der Abläufe erfolgt in UML über die Modellierung von Aktivitäten oder dem Nachrichtenaustausch zwischen den Objekten. UML bietet dabei unterschiedliche Modellierungsmethodiken an, wie Aktivitätsdiagramme, die Abfolgen von Aktivitäten beschreiben, oder Sequenzdiagramme, die ein konkretes Szenario anhand von Nachrichten, welche die beteiligten Objekte untereinander austauschen, definieren. Kommunikationsdiagramme modellieren schließlich alle Nachrichten, die zwischen den Objekten ausgetauscht werden. Mit Interaktionsübersichtsdiagrammen bietet UML2 auch eine Methodik an, die beide Ansätze (Modellierung von Aktivitäten und Nachrichten) verbindet. Aktivitätsdiagramme bilden dabei den Rahmen der Modellierung, wobei die einzelnen Aktivitäten durch Sequenzdiagramme verfeinert werden können.

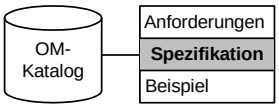
Zustandsdiagramme bedienen sich schließlich endlicher Automaten für die Modellierung von Systemzuständen, die durch bestimmte Events in einen anderen Zustand überführt werden.

Die in UML vorgestellten Modelle sind dabei speziell auf die Modellierung dynamischer Aspekte von Software und die Modellierung von Geschäftsprozessen ausgelegt. Für beide Zielsetzungen werden in UML mächtige Werkzeuge angeboten.

Für die Modellierung von Interaktionen mit dem metaphernbasierten Ansatz bieten die Modelle nur unzureichende Unterstützung:

- Aktivitätsdiagramme modellieren ausschließlich Aktivitäten und deren Abfolge. Damit lassen sich Interaktionsmetaphern und deren Abfolgen modellieren, ein Gegenstück zu Objektmetaphern ist allerdings nicht vorgesehen. Auch die Modellierung der Interaktionsmöglichkeiten über die Sichtbarkeit von Objekten kann nicht modelliert werden.
- Sequenzdiagramme und Zeitverlaufsdiagramme bilden nur einen Teil der Interaktion ab, da Sequenzdiagramme einen definierten Durchlauf beschreiben, während Zeitverlaufsdiagramme Eigenschaften von Objekten in Beziehung zueinander setzen. Daher verfügen beide nicht über eine ausreichende Mächtigkeit für die geforderte Modellierung.

[27] Anwendungsfalldiagramme gehören erst seit der UML Version 2.0 zu den Verhaltensmodellen.



- Kommunikationsdiagramme bieten Modellelemente für Objekte und Nachrichten an, die als Interaktionen interpretiert werden können. Sie bieten aber keine Konstrukte zur Modellierung von Zuständen. In einem Kommunikationsdiagramm sind immer alle Objekte verfügbar und prinzipiell auch alle Nachrichten zwischen den Objekten möglich. Zur Definition von Abfolgen von Nachrichten ist lediglich eine einfache sequentielle Struktur vorgesehen.
- Zustandsdiagramme können für die Modellierung eingesetzt werden, da ein Zustand die Menge der aktuell verfügbaren Interaktionen beschreiben könnte. Durch eine Interaktion würde ein anderer Zustand erreicht werden, der sich durch eine andere Menge von Objektmetaphern auszeichnen würde. Eine Modellierung mittels Zustandsdiagrammen würde aber eine exponentielle Anzahl an Zuständen erfordern, da n mögliche Objektmetaphern zu 2^n möglichen Zuständen führen können. Auch wenn nicht alle möglichen Zustände sinnvoll sind, ist dennoch bereits für kleine Modelle eine Grenze erreicht, die sich nicht mehr bewältigen lässt.

Absichtlich außen vor gelassen wurden zusätzliche Eigenschaften von Modellen in UML, wie die Komposition von Modellelementen und die Integration von Ablaufstrukturen in die Modellierung. Bereits die grundlegenden Anforderungen für eine Modellierung auf Basis von Objektmetaphern können nicht mit den UML-eigenen Diagrammen umgesetzt werden.

5.2.4.2 Dialogspezifikationssprachen

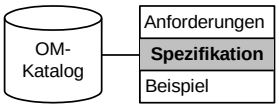
Dialogspezifikationssprachen bilden eine zweite Klasse von Modellen, die für die Beschreibung von Interaktionen verwendet werden können. Hier kann zwischen mehreren Modellspezifikationen unterschieden werden.

Zustandsübergangsdiagramme

Zustandsübergangsdiagramme wurden bereits in Abschnitt 5.1.4.1 als typische Vertreter von Dialogspezifikationssprachen eingeführt. Die Nutzung von Zustandsübergangsdiagrammen für die Modellierung von Interaktionen mit dem metaphernbasierten Ansatz wurde im Rahmen der Betrachtung von UML (dort Zustandsdiagramme genannt) in Abschnitt 5.2.4.1 diskutiert.

Statecharts

Statecharts unterstützen eine formale Beschreibung reaktiver Systeme, die kontinuierlich auf externe und interne Events reagieren, beispielsweise Telefonvermittlungen, Kommunikationssysteme, Betriebssysteme und auch Benutzerschnittstellen. Mathematische Grundlagen von Statecharts sind die sogenannten Higraphen [Har88]. Statecharts sind eine Erweiterung von Zustandsübergangsdiagrammen, die auf Higraphen basieren. Sie besitzen gegenüber Zustandsübergangsdiagrammen einen erweiterten Zustandsbegriff, da Zustände durch die Definition von Unterzuständen komplex strukturiert werden können. Zwischen den Unterzuständen eines komplex strukturierten Zustands können Zustandsübergänge existieren. Komplexe Zustände können über XOR oder AND verknüpft werden. Über die AND-Verknüpfung von Zuständen



kann die Nebenläufigkeit des Systems modelliert werden, während eine XOR-Verknüpfung lediglich die Aktivierung eines Unterzustandes in einem komplexen Zustand erlaubt.

Zustandsübergänge können mit Events, Prädikaten und Aktionen markiert werden. Wie bei den Zustandsübergangsdiagrammen lösen Events einen Zustandsübergang aus, wenn das Prädikat erfüllt ist. Aktionen können selbst wieder Events erzeugen, die an alle aktiven Zustände verteilt werden können und damit eine interne Kommunikation auslösen.

Statecharts beheben somit die Probleme der Zustandsübergangsdiagramme, indem sie inhärent Nebenläufigkeit, eine hierarchische Strukturierung von Zuständen und eine Event-Verteilung erlauben.

Auch wenn Statecharts eine Vielzahl von Mechanismen besitzen, welche die Anzahl der für die Modellierung notwendigen Zustände reduzieren, bleibt beim metaphernbasierten Ansatz die Problemstellung erhalten. Ein Zustand ist demnach weiterhin eine Menge an verfügbaren Objektmetaphern. Eine hierarchische Strukturierung ist dabei nur schwer realisierbar, so dass die zusätzlichen Mechanismen von Statecharts gegenüber den Zustandsübergangsdiagrammen nicht zum Tragen kommen.

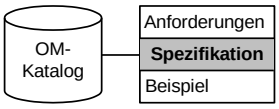
Constraints

Constraints gehören zur Klasse der deklarativen Dialogspezifikationsmethoden, die insbesondere die Abhängigkeit zwischen den Daten der Benutzerschnittstelle und der Applikation, sowie die Datenabhängigkeiten innerhalb der Benutzerschnittstelle in den Mittelpunkt stellen. Deklarative Dialogspezifikationsmethoden beschreiben den Dialogablauf nicht explizit, sondern durch die zwischen zulässigen Dialogzuständen auszuführenden Operationen oder die notwendigen Eigenschaften zulässiger Dialogzustände. Constraints spezifizieren die einzuhaltenden Beziehungen zwischen Attributen von Präsentations-, Dialogkontroll- und Applikationsobjekten. Das Verhalten einer durch Constraints beschriebenen Benutzerschnittstelle wird durch einen sogenannten „Constraint-Solver“ gesteuert. Bei Verletzung eines Constraints als Folge von Nutzeraktionen steuert er die Wiederherstellung eines konsistenten Zustandes durch den Aufruf von Methoden, die unter anderem die Benutzeroberfläche aktualisieren und die Applikation informieren.

Constraints sind nicht in der Lage als Modellierungsspezifikation zu fungieren. Vielmehr ermöglichen sie die Spezifikation und den Erhalt eines definierten Verhaltens.

Interaktionsdiagramme (IAD) nach [Den91]

Interaktionsdiagramme (kurz IAD) sind eine Variante von Zustandsübergangsdiagrammen. Ein IAD ist eine formale Spezifikation eines Dialogablaufes und kann zur Laufzeit von der Dialogsteuerung interpretiert werden, d.h. es ist automatisch ausführbar. Im Unterschied zu herkömmlichen endlichen Automaten verfügen IADs über zwei Arten von Stellen (Zustände und Aktionen) und zwei Arten von Zustandsübergängen (Ereignisse und Aktionsergebnisse). Befindet sich das System in einem Zustand, so wartet es auf eine Benutzeraktion (Ereignis). Aktionen dagegen sind vom System ausführbare Funktionen. Zwischen



zwei Zuständen kann es beliebig viele Aktionen geben. IADs modellieren somit neben Nutzerinteraktionen auch die Aktionen des Systems.

Für diese Modellierung existiert ein Framework, das auf JavaStruts (vgl. [Weß06]) und auf anderen modernen Web-Techniken (Servlets, JSPs, ApplicationServer) basiert, und sich im Produktivbetrieb befindet.

Interaktionsdiagrammen liegt die gleiche Problemstellung zugrunde, die in Zustandsübergangsdiagrammen und Statecharts zu finden ist, da auch diese auf definierten Zuständen arbeiten.

Ereignismodelle

Das Ereignismodell hat sich aufgrund der engen Verwandtschaft mit den heutigen verbreiteten Fenstersystemen im Bereich der kommerziellen User Interface Management Systeme weitgehend durchgesetzt. Dialoge werden dabei durch eine Menge gleichzeitig aktiver Module für die Event-Verarbeitung, den so genannten Eventhandlern, spezifiziert. Sie besitzen lokale Variabeln für die Verwaltung ihres Zustandes und lokale Prozeduren für die Verarbeitung bestimmter, vorgegebener Events. Die Dialogspezifikation durch Eventhandler impliziert eine verteilte Dialogablaufsteuerung und eignet sich insbesondere für die Beschreibung von Benutzerschnittstellen mit gleichzeitig aktiven Teildialogen und vom Dialogzustand unabhängigen Interaktionsmöglichkeiten.

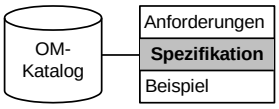
Der Dialogzustand ergibt sich aus der Kombination der lokalen Zustände der Eventhandler und besitzt eine explizite Repräsentation. Wesentlicher Nachteil der Dialogspezifikation ist das Fehlen einer expliziten Dialogablaufrepräsentation, wie sie in Zustandsübergangsdiagrammen zu finden ist. Aus diesem Grund existiert auch keine allgemeingültige Notation. Eventhandler beschreiben die Verarbeitung von Events kontextunabhängig. Abhängigkeiten vom Dialogablauf können nur durch Kommunikation zwischen Eventhandlern und der lokalen Verwaltung von Zustandsinformationen programmiert, aber nicht spezifiziert werden.

Ereignismodelle eignen sich als Ausgangslage für die geforderte Modellierung, da dem metaphernbasierten Ansatz nur indirekt eine Ablaufsteuerung zugrunde liegt. Dennoch erfordern die Beziehungen zwischen den Objektmetaphern, die über Interaktionen erzeugt oder zerstört werden, eine gewisse Spezifikation. Eventmodelle können dieses allerdings nicht leisten.

Zustandsbäume

Eine Erweiterung der Ereignismodelle um die Strukturierungsmöglichkeiten der Zustandsübergangsdiagramme stellen Zustandsbäume [Rum88] dar. Sie definieren den Ablauf durch hierarchisch angeordnete Zustände, denen lokale Variablen, Aktionen und Eventhandler zugeordnet werden.

Da sich eine hierarchische Dekomposition verfügbarer Objektmetaphern nur schwer realisieren lässt, ist auch diese Methodik nicht vielversprechend.



Graph-Grammatiken

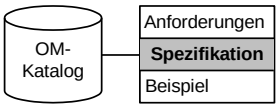
Im Vergleich zu den bisher vorgestellten Modellierungsansätzen bieten Graph-Grammatiken (Graph Grammers im Englischen) [Suc96] eine gänzlich andere Form der Dialogspezifikation. Sie basieren auf der Beobachtung, dass sich graphische Benutzerschnittstellen durch Dialogstati und Dialogstatustransformationen auszeichnen. Ein Dialogstatus repräsentiert die Darstellung der Interaktionsobjekte einer Anwendung in einer bestimmten Situation. Er kann formal durch einen gerichteten attributierten Graphen beschrieben werden. Knoten repräsentieren dabei Interaktionsobjekte und Kanten die Beziehungen zwischen den Interaktionsobjekten und der unterliegenden Applikation. Statuswechsel, ausgelöst durch ein Event (entweder eine Nutzerinteraktion oder ein Systemevent), werden durch so genannte „graph rewrite rules“ beschrieben. „Graph rewrite rules“ beschreiben auf einer formalen Ebene die Transformation des existierenden Graphen.

Die Darstellung von Objektmetaphern als gerichteter Graph spiegelt die Idee des in Abschnitt 5.2.3.1 vorgestellten Metamodells für die Interaktionsmodellierung mit Hilfe von Metaphern wieder. Die Sichtbarkeit von Objektmetaphern kann in solch einem Graphen abgebildet werden. Die Knoten des Graphen können als Objektmetaphern und die Kanten als Beziehungen zwischen den Objektmetaphern interpretiert werden. Interaktionen lösen nun eine Transformation des Graphen in einen anderen aus, der die verfügbaren Elemente nach der Interaktion beschreibt. Einziges Manko ist die Modellierung von Interaktionsmöglichkeiten in einem bestimmten Status. Anhand des Modells ist nicht direkt ersichtlich, welche Interaktionen in einem definierten Status möglich sind. Hierzu müssen die „graph rewrite rules“ daraufhin analysiert werden, ob sie von einem gegebenen Status heraus ausgelöst werden können. Dennoch ist die Modellierung von Interaktionen mit Objektmetaphern durchaus mit Graph Grammers realisierbar.

Petri-Netze

Grundsätzlich erfolgt die Verwendung von Petri-Netzen (vgl. [WL08]) als Dialogmodell analog zu Zustandsübergangsdiagrammen. Im Unterschied zu Zustandsübergangsdiagrammen können durch Verzweigungen an den Transitionen mehrere Stellen (Zustände) gleichzeitig aktiv werden. Dadurch kann Parallelität direkt ausgedrückt werden. Man unterscheidet bei der Verwendung von Petri-Netzen für die Dialogmodellierung zwischen zwei Interpretationen: einerseits werden so genannte Event-Transitionen eingeführt, die als Schnittstelle deklariert werden und durch externe Events ausgelöst werden. Andererseits werden Stellen als Event-Stellen definiert, die aufgrund von externen Events mit Marken belegt werden. Beiden Interpretationen gemein ist, dass Stellen und Transitionen als komplex gekennzeichnet werden können. Komplexe Stellen (und Transitionen) werden wiederum über ein Petri-Netz beschrieben.

Petri-Netze stellen einen vielversprechenden Ansatz für die angestrebte Modellierungsmethodik dar. Stellen können Objektmetaphern repräsentieren und Transitionen Interaktionsmetaphern. Die Kanten können die Beziehungen zwischen Objekt- und Interaktionsmetapher abbilden. Schließlich bietet die Möglichkeit zur Markierung von Stellen ein Werkzeug für die Modellierung von Sichtbarkeiten von Objektmetaphern: markierte Stellen gelten als sichtbar. In einem Petri-Netz definiert die Markierung, welche



Transitionen aktiviert werden. Übertragen auf die Dialogmodellierung bedeutet dies, dass Interaktionsmöglichkeiten bestehen, wenn die entsprechenden Objektmetaphern sichtbar sind. Solch eine Logik wird für die Interaktionsmodellierung mit Hilfe von Metaphern benötigt.

Dialognetze nach Janssen [BFJ96]

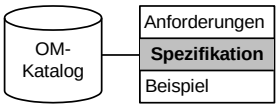
Dialognetze werden als integriertes Beschreibungskonzept für Dialogabläufe bei graphischen Benutzersystemen verwendet. Sie basieren auf beschrifteten Bedingungs-/Ereignisnetzen (Petri-Netzen) und beschreiben Dialogabläufe auf Fensterebene. Die Stellen der Dialognetze repräsentieren Fenster, und Transitionen die möglichen Interaktionen (z.B. Button, Menüs, Verweise, usw.), wenn ein Fenster sichtbar ist. Eine markierte Stelle modelliert die Sichtbarkeit des Fensters, das durch die Stelle repräsentiert wird. Jede Marke in einer Stelle wird dabei als eine Instanz eines Fensters aufgefasst. Markierungsregeln können dazu benutzt werden, um im Modell darzustellen, welche Interaktionen in einem bestimmten Zustand des Dialognetzes möglich sind. Besitzt beispielsweise eine Transition mehr als eine Eingangsstelle, so bedeutet dies auch, dass mehr als ein Fenster sichtbar sein muss, damit die Interaktion ausgeführt werden kann. Zusätzlich wird eine Beschreibung auf Objektebene mittels Constraints verwendet. Die Objektebene beschreibt Oberflächenobjekte (Tabellen, Auswahllisten, Eingabefelder, usw.) der Fenster und die dynamischen Beziehungen zwischen Oberflächenobjekten untereinander. Constraints werden in Dialognetzen in Form einer „Pseudosprache“ beschrieben und definieren Regeln, wie beispielsweise, dass erst nach dem Ausfüllen eines Formulars der „Absenden-Button“ aktiviert wird.

Wie Petri-Netze sind auch Dialognetze für die angestrebte Modellierung anwendbar. Gegenüber den Petri-Netzen bieten Dialognetze zusätzlich den Vorteil, dass sie über Erweiterungen (vgl. Abschnitt 5.3) verfügen, die speziell für die Modellierung von Interaktionen ausgelegt sind.

5.2.4.3 Zusammenfassung und Resümee

Die Betrachtung existierender Modellierungsansätze für die Interaktionsmodellierung zeigt, dass obwohl eine große Anzahl von Modellierungsmethodiken existieren, viele Methodiken primär die Modellierung von Abläufen als Zielsetzung verfolgen. Insbesondere am Beispiel von UML2 ist ersichtlich, dass die Modellierung von Prozessen, aber natürlich auch die Modellierung des Verhaltens von Software-Systemen im Fokus stehen. Die Modellierung von Interaktionen, wie sie in dieser Arbeit definiert sind, erfolgt in erster Linie über die Sichtbarkeit von Objektmetaphern. Abläufe werden somit nur indirekt modelliert, indem die Menge von möglichen Interaktionen durch sichtbare Objektmetaphern bestimmt wird, und Interaktionen die Sichtbarkeit ändern können. Auch die Betrachtung von Dialogspezifikationssprachen zeigt, dass ein Großteil der Modelle Abläufe beschreibt, beziehungsweise ergänzende Ansätze wie Constraint- und Event-Modelle nicht über ausreichend formale Mächtigkeit zur Beschreibung von Interaktionen verfügen.

Das Resultat der Betrachtung ist, dass Modelle, die auf Graph-Grammatiken, Petri-Netzen und insbesondere Dialognetzen basieren, für eine Modellierung verwendet werden können. Auf eine eingehende Ge-



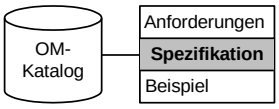
genüberstellung von Graph Grammers und Petri-Netzen wird an dieser Stelle verzichtet, da Petri-Netze die gestellten Anforderungen bereits vollständig erfüllen. Der Graph Grammer Ansatz hingegen benötigt eine genauere Betrachtung und auch eine Überarbeitung, damit für eine Analyse von Interaktionsmöglichkeiten in einem gewissen Zustand nicht alle „graph rewrite rules“ betrachtet werden müssen. Anstatt Petri-Netze direkt als Ausgangsbasis für eine Modellierungsmethodik zu wählen, werden Dialognetze herangezogen. Diese verfügen bereits über Eigenschaften für eine Modellierung von Interaktionen, die man sonst erneut auf Petri-Netzen definieren müsste.

5.3 Interaktionsmodellierung mit Abstract Interaction Nets (AIN)

Das in Abschnitt 5.2.3.1 eingeführte Metamodell für die Interaktionsmodellierung mit Hilfe eines Metaphernansatzes beschreibt die grundlegenden Modellierungselemente und Konzepte der Modellierung. Eine Methodik für eine Modellierungsspezifikation, die es erlaubt, auf einer formalen Ebene Interaktionen zu beschreiben, ließ sich nicht direkt aus dem Metamodell ableiten. Die Betrachtung von existierenden Ansätzen (in Abschnitt 5.2.4) für die Interaktionsmodellierung hat aber gezeigt, dass insbesondere Dialognetze nach Janssen (vgl. [Jan93]) eine geeignete Grundlage bilden können. In diesem Kapitel wird die Modellierungsmethodik Abstract Interaction Nets (kurz AIN) erarbeitet. AIN stellen eine formale Beschreibungsmethodik bereit, die es dem Modellierer erlaubt, Interaktionen mit Hilfe von Metaphern zu modellieren. Sie werden insbesondere als Ausgangslage für eine automatisierte Generierung von Benutzerschnittstellen auf der Basis einer abstrakten Spezifikation herangezogen, bieten sich aber auch als Werkzeug für Anforderungsanalysen von Benutzerschnittstellen an.

Als Grundlage für die Notation werden Dialognetze nach Janssen herangezogen. Dialognetze sind für die Modellierung von Dialogabläufen auf Fensterebene entwickelt worden, die insbesondere die Parallelität der Abläufe unterstützen. Weiterhin verfügen Dialognetze über Konstrukte zur Vereinfachung und Reduktion der Modellgröße. Dialognetze nutzen selbst beschriftete Bedingungs- und Ereignisnetze (B/E-Netze) und bieten weiterhin die Möglichkeit zur Definition von Constraints für die Beschreibung des Verhaltens innerhalb eines Fensters. Für eine eingehende Beschreibung von Dialognetzen sei auf [BFJ96] verwiesen.

Im Folgenden wird die Notation AIN für die Modellierung von abstrakten Interaktionen erarbeitet. Im Einzelnen werden die Konstrukte (Stellen, Transitionen und Relationen) eines AINs vorgestellt und anschließend diskutiert, wie das Pendant des Konstruktes in Dialognetzen realisiert wurde und welche Konzepte übernommen, beziehungsweise geändert werden mussten. In einem ersten Schritt werden die Grundlagen der Modellierungsmethodik eingeführt, und anschließend die wichtigen Erweiterungen vorgestellt. Im Einzelnen sind dies: „komplexe Transitionen“ für die Beschreibung von Interaktionen, die mehrere Schritte kapseln; die Modellierung von „Systeminteraktionen“ für die Beschreibung von Interaktionen, die abhängig von Berechnungs- oder Retrieval-Funktionalitäten des Informationssystems sind.



5.3.1 Grundlagen von AIN

Nur die Grundform von Dialognetzen ist für die Interaktionsbeschreibung von Bedeutung, da viele der zusätzlichen Konstrukte wie „Optionale Flüsse“, „Modale Stellen“, „voll spezifizierte Dialognetze“ und auch die Definition von „Constraints“ für eine ausführbare Dialogbeschreibung sinnvolle Ergänzungen darstellen, für eine Interaktionsbeschreibung aber nicht benötigt werden.

Analog zu Dialognetzen ist ein AIN wie folgt definiert:

Ein Abstract Interaction Net ist ein 7-Tupel AIN = (S, T, F, K, W, B, m0).

Hierbei ist S eine Menge von Stellen, T eine Menge von Transitionen, F eine Flussrelation und es gilt S∩ T = ∅ , und F ⊆ (SxT)∪ (TxS).

K ist eine Kapazitätsfunktion für die gilt K: S --> {0,∞ }.

W eine Gewichtsfunktion mit W: F--> ℕ.

m0 ist die Startmarkierung des Netzes mit m0: S--> ℕ.

**t ={s|(s, t)∈ F}* *die Menge der Eingangsstellen von t,*

t ={s|(t, s) ∈ F}* *die Menge der Ausgangsstellen von t,*

**t∩ t* heißt die Menge der Nebenstellen von t.*

B ist eine Menge von Beschriftungen mit b: S∪ T --> B.

Die aktuelle Markierung m: S --> ℕ bezeichnet den Zustand des Netzes und m(s) ist die Anzahl der Marken von s.

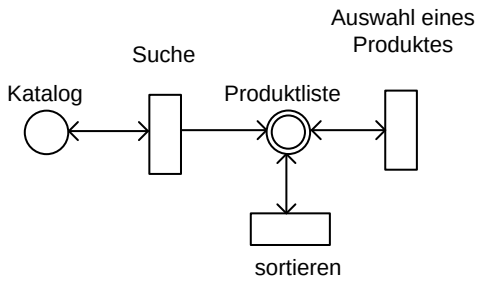


Abbildung 21: Graphische Repräsentation eines AIN

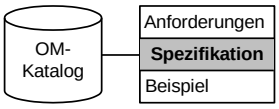


Abbildung 21 demonstriert die graphische Repräsentation eines AIN. Wie in Petri-Netzen üblich, werden Stellen durch Kreise, Transitionen durch Rechtecke und Flüsse durch gerichtete Kanten repräsentiert. Eingehende Flüsse in eine Transition markieren die Eingangsstellen und ausgehende die Ausgangsstellen. Nebenstellen werden mit einem Doppelpfeil gekennzeichnet. Da Stellen eines AIN entweder eine Kapazität von 1 oder unendlich haben, wird auf eine numerische Kennzeichnung verzichtet. Stattdessen werden Stellen mit einer Kapazität von 1 als einfache Kreise gekennzeichnet, während Stellen mit einer unendlichen Kapazität durch umrandete Kreise repräsentiert werden. Flussrelationen ohne numerische Kennzeichnung haben standardmäßig ein Gewicht von 1. Wie in Dialognetzen dient die Beschriftung der Stellen und Transitionen der Kennzeichnung dieser Elemente. m0 definiert die anfänglich verfügbaren Objektmetaphern. Es ist dabei anzumerken, dass Dialognetze auf eine Startmarkierung verzichten und stattdessen ausgezeichnete Transitionen nutzen, die automatisch bei der Initialisierung des Netzes schalten, und damit eine Startmarkierung erstellen. Auf eine graphische Repräsentation von m0 wird verzichtet.

In Dialognetzen werden Stellen als Fenster einer Benutzerschnittstelle interpretiert, deren Markierung definiert, ob das Fenster sichtbar, also für den Benutzer verfügbar ist. Transitionen definieren mögliche Interaktionen des Benutzers innerhalb des Fensters. Die Verfügbarkeit (Markierung der Stelle) eines Fensters definiert mögliche Interaktionen des Benutzers. In einem AIN werden Stellen als Objektmetaphern und Transitionen als Interaktionsmetaphern definiert. Eine markierte Stelle modelliert die Verfügbarkeit einer Objektmetapher. Stellen mit einer Kapazität von 1 modellieren dabei Objektmetaphern, von denen nur eine Instanz existieren darf, während Stellen mit einer Kapazität von unendlich beliebig viele Instanzen einer Objektmetapher modellieren. Die Anzahl der Markierungen gibt dabei Auskunft über die Menge der für den Benutzer verfügbaren Objektmetaphern. In Transitionen eingehende Kanten modellieren die für die Interaktion (Interaktionsmetapher) notwendigen Objektmetaphern, während ausgehende Kanten die Objektmetaphern bestimmen, die durch die Interaktionsmetapher erzeugt werden. Nebenstellen (in Abbildung 21 ist die „Produktliste“ eine Nebenstelle der Transition „sortieren“) einer Transition modellieren die Manipulation von Objektmetaphern, die sich nicht direkt auf die Verfügbarkeit auswirkt. Für Nebenstellen gilt dabei, dass die Gewichtsfunktion der eingehenden Kante gleich der ausgehenden Kante ist. Formal heißt das:

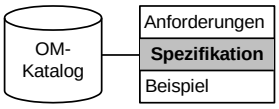
Für alle Nebenstellen p einer Transition t gilt:

$p \in {}^*t \cap t^*: W(p,t)=W(t,p)$

Die Beschreibung der Interaktionsmöglichkeiten erfolgt über die Verfügbarkeit der Objektmetaphern. In einem AIN wird die Verfügbarkeit über die Markierung von Stellen modelliert. In einem Zustand mögliche Interaktionen werden über aktivierte Transitionen modelliert. Es gilt:

$t \in T$ heißt aktiviert in einer Markierung m, wenn gilt:

(1) $\forall p \in {}^*t: m(p) \geq W(p,t)$



Ist eine Transition t aktiviert, kann diese Transition schalten. Wenn sie schaltet, wird für alle Stellen die Anzahl der Marken wie folgt berechnet:

*(1) m'(p) = m(p) - W(p,t) falls p∈ *t und p ∉ t**

*(2) m'(p) = m(p) + W(t,p) falls p ∉ *t und p ∈ t* und K(t,p)>1*

*(3) m'(p) = 1 falls p ∉ *t und p ∈ t* und K(t,p)=1*

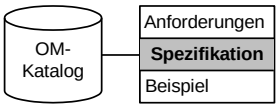
(4) m'(p) = m(p) falls p eine Nebenstelle von t ist.

(5) m'(p) = m(p) sonst

Aktivierte Transitionen in einem AIN beschreiben Interaktionen, die der Benutzer durchführen kann. Das Schalten einer Transition beschreibt, dass der Benutzer diese Interaktion durchgeführt hat. Welche der aktivierten Transitionen tatsächlich schaltet, ist von der Interaktion des Benutzers abhängig und kann nicht vorhergesagt werden.

Das Gewicht der Flussrelationen beschreibt für Eingangsstellen, wie viele der Objekte für die Interaktion notwendig sind und für Ausgangsstellen, wie viele Instanzen der Objektmetapher erzeugt werden. Ausgangsstellen mit einer Kapazität von 1 modellieren, dass die Interaktion zwar durchgeführt werden kann, die Markierung der Ausgangsstelle aber überschrieben wird. Transitionen, die Beziehungen zu Nebenstellen haben, verändern nicht die Anzahl der Marken in den Nebenstellen, da eine Nebenstelle sowohl eine eingehende, als auch eine ausgehende Kante in sich vereinigt. So können Nebenstellen notwendige Objektmetaphern für die Interaktion beschreiben, ohne dass diese beim Schalten der Transition vernichtet werden, da laut Definition die Gewichtsfunktion für die Eingangsstelle gleich der Ausgangsstelle sein muss. Wie in Dialognetzen modelliert das Schalten von Transitionen, die über Nebenstellen verfügen, die Manipulation der entsprechenden Objektmetaphern der Nebenstellen.

Einer der wesentlichen Unterschiede von AIN zu Dialognetzen ist, dass Transitionen unabhängig von ihren Ausgangsstellen schalten dürfen. Die Reduzierung der Auswahl für mögliche Kapazitäten der Stellen auf entweder 1 oder unendlich ermöglicht sowohl die Modellierung, dass von bestimmten Objektmetaphern nur eine, als auch dass beliebig viele Instanzen existieren dürfen. AINs sind durch die „Kapazität unendlich“ in der Lage, nicht nur unterschiedliche Instanzen einer Objektmetapher zu modellieren, sondern weiterhin Interaktionen abzubilden, die mehr als nur eine Instanz einer Objektmetapher zur Durchführung benötigen. Insbesondere ermöglicht dieser Ansatz die wiederholte Ausführung von Interaktionen, sobald die benötigten Interaktionsobjekte verfügbar sind. Damit folgen AINs dem Leitbild der metaphernbasierten Modellierung, in dem der „Normalfall“ vorsieht, dass Interaktionen möglich werden, sobald die benötigten Objekte verfügbar sind. In Dialognetzen hingegen unterbindet die Belegung von Ausgangsstellen einer Transition deren wiederholte Ausführung.



In Dialognetzen ist die wiederholte Durchführung von Interaktionen ein Sonderfall, für dessen Modellierung drei erweiternde Konstrukte (optionale Flüsse, komplexe Stellen und dynamische Teildialoge) von Dialognetzen benötigt werden.

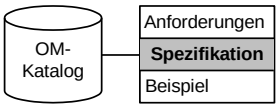
Optionale Flüsse beschreiben dabei Flussrelationen, die keine Bedingungen an das Schaltverhalten stellen, sich aber beim Schalten der Transition, falls es die Markierungsregeln erlauben, entsprechend verhalten. Primär wird dieses Konstrukt verwendet, um die Komplexität von Dialognetzen zu reduzieren. Indem Ausgangsstellen von Transitionen mit optionalen Flüssen verbunden werden, kann die Transition schalten, auch wenn die Stelle besetzt ist. Dieses Konstrukt ermöglicht somit das wiederholte Schalten einer Transition, es modelliert aber nicht, dass unterschiedliche Instanzen der Stelle durch die Interaktion geschaffen werden. Zur Modellierung dieses Sachverhaltes werden in Dialognetzen zwei weitere Konstrukte benötigt.

Komplexe Stellen bieten in Dialognetzen die Möglichkeit, wieder verwendbare Module zu schaffen, indem einer Stelle ein so genanntes Unternetz zugeordnet wird, das selbst ein Dialognetz ist. Die Beziehungen zwischen Dialognetz und Unternetz sind dabei einfach gehalten. Wird im Dialognetz von einer Stelle eine Marke entfernt, so werden alle Marken aus dem Unternetz der Stelle entfernt. Sind in einem Unternetz alle Marken entfernt worden, so wird die Markierung der entsprechenden Stelle im Dialognetz entfernt. Anzumerken ist weiterhin, dass komplexe Stellen mehrfach in einem Dialognetz verwendet werden dürfen.

Dynamische Teildialoge modellieren, dass komplexe Teildialoge (also komplexe Stellen) als unterschiedliche Instanzen geöffnet werden. Die Erzeugung unterschiedlicher Instanzen wird in Dialognetzen durch komplexe Stellen modelliert, die als Eingangskante eine optionale Kante besitzen. Damit wird modelliert, dass die Transition mehrfach schalten kann (aufgrund der optionalen Stelle), und in jedem Schaltschritt eine Instanz der komplexen Stelle erzeugt wird.

Auf eine Adaption dieser drei Konstrukte für AINs wurde aus zwei Gründen verzichtet: erstens sollte die wiederholte Ausführung von Interaktionen mit all ihren Folgen, die insbesondere die Erzeugung von neuen Instanzen von Objektmetaphern mit einbezieht, ein natürlicher Teil der Modellierung sein; zweitens kann das Konstrukt der komplexen Stellen nicht dem Anspruch der Interaktionsmodellierung gerecht werden, nämlich dass Interaktionen spezifiziert werden müssen. Über Objektmetaphern (die Stellen in AINs) wird lediglich modelliert, welche Interaktionen in einem gegebenen Zustand durchgeführt werden können. Interaktionen hingegen können komplex sein und bedürfen daher eines Konstruktes, das eine Verfeinerung ermöglicht.

Eine letzte Erweiterung, die von Dialognetzen auf AINs übertragen wurde, ist die inhärente Modellierung von „Standard Interaktionen“, wie die Möglichkeit Fenster zu minimieren, zu schließen bzw. in ihrer Größe zu verändern. Diese Operationen werden in Dialognetzen jedem Fenster zugeschrieben, ohne dass diese Möglichkeiten tatsächlich modelliert werden müssen. In AINs können Objektmetaphern (auch Instanzen einer Objektmetapher) entsprechend jederzeit zerstört werden.



Im Folgenden werden drei Erweiterungen für AINs vorgestellt, die notwendige Konstrukte für eine Interaktionsmodellierung repräsentieren und keine direkte Entsprechung in Dialognetzen finden. Im Einzelnen sind dies:

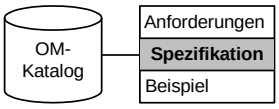
- Die bereits angesprochenen komplexen Transitionen, die eine hierarchische Strukturierung von Interaktionen ermöglichen.
- Systeminteraktionen zur Modellierung des dynamischen Systemverhaltens und ihren Auswirkungen auf die Interaktionsmöglichkeiten des Benutzers. Systeminteraktionen modellieren dabei den Zugriff auf dynamische Datenquellen oder Dienste, die außerhalb der Interaktionsmodellierung stehen, aber Auswirkungen auf diese haben.
- Konzepte zur Wiederverwendung von Modellfragmenten. Insbesondere wird die Modellierung von Interaktionen mit Hilfe eines Katalogs von Objekt- und Interaktionsmetaphern vorgestellt. Katalogisierte Objekte verfügen zusätzlich über eine über die Interaktionsmodellierung hinausgehende Beschreibung, die für eine automatisierte Generierung von Benutzerschnittstellen benötigt wird.

5.3.2 Komplexe Transitionen

Komplexe Transitionen ermöglichen die Kapselung von komplexen Interaktionsabläufen, indem Unternetze, die ebenfalls AINs sind, für die Beschreibung dieser Interaktionen verwendet werden. Das Schalten einer komplexen Transition hat zur Folge, dass das Unternetz initialisiert wird und für den Benutzer zur Verfügung steht. Schaltet eine komplexe Transition mehrfach, so werden unterschiedliche Instanzen des Unternetzes erzeugt. Das Schalten einer komplexen Transition eröffnet dabei für den Benutzer einen neuen „Unter-Interaktionsraum“ (vgl. Abschnitt 5.2.3.3), der eigene Objekt- und Interaktionsmetaphern enthalten kann. Dabei sind Unternetze eng mit dem übergeordneten AIN verknüpft, da alle Eingangs- und Ausgangsstellen der komplexen Transition Teile beider Netze sind. Eine komplexe Interaktion gilt als abgeschlossen, sobald durch Interaktionen im Unternetz mindestens eine Ausgangsstelle der komplexen Transition eine Markierung erhalten hat. Ein Unternetz kann jederzeit durch den Benutzer zerstört werden. Dies gilt als inhärente Eigenschaft von Unternetzen und braucht nicht modelliert zu werden.

Formal gilt:

> *Ein AIN = (S, T, F, K, W, B, m0, Tk) heißt AIN mit komplexen Transitionen, wobei Tk die Menge der komplexen Transitionen ist. Es gilt* $Tk \subseteq T$.
>
> *taus* $\subseteq S$ *beschreibt die Menge aller Ausgangsstellen einer komplexen Transition* $t \in Tk$ *mit* $taus = \{s \in S \mid (t, s) \in F\}$
>
> *tein* $\subseteq S$ *beschreibt die Menge aller Eingangsstellen einer komplexen Transition* $t \in Tk$ *mit* $tein = \{s \in S \mid (s, t) \in F\}$



Ein Unternetz einer Transition ist definiert als:

> *Jeder komplexen Transition t ∈ Tk wird ein Unternetz AIN (t) = (S(t), T(t), F(t), K(t), W(t), B(t), m0(t), Tk(t), Taus(t), Tein(t)) zugeordnet.*
>
> *Dabei gilt taus ∪ tein = S(t)∩ S, T(t)∩ T=ø, F(t)∩ F=ø, K(t)∩ K=ø, W(t)∩ W=ø, B(t)∩ B=ø und m0(t):S*(t) --> ℕ mit S*={s|s∈ S(t) }*

Laut Definition sind die Eingangs- und Ausgangsstellen einer komplexen Transition t, sowohl Teil eines AINs, als auch Teil des Unternetzes AIN(t). Dass heißt aber auch, dass tein und taus notwendigerweise Teile des Unternetzes von AIN sind. Die Beziehungen eines Unternetzes zum übergeordneten Netz beschränken sich ausschließlich auf die Eingangs- und Ausgangsstellen einer Transition.

Weiterhin können Unternetze selbst wieder komplexe Stellen enthalten. Mit diesem Konstrukt ist es möglich, beliebig tiefe Hierarchien von Interaktionen zu modellieren. Um Rekursionen im Netz zu verhindern, muss sichergestellt werden, dass die komplexe Transition selbst nicht für ihre eigene Beschreibung herangezogen wird. Dies würde zu rekursiven Strukturen im Netz führen, die bei der Modellierung von Interaktionen zu vermeiden sind.

Das Schalten einer komplexen Transition initiiert das Unternetz dieser Transition mit der entsprechenden Anfangsmarkierung m0(t). Dem Benutzer stehen dann die Objektmetaphern des Unternetzes zur Verfügung, mit denen er interagieren kann. Der Schaltvorgang der komplexen Transition gilt erst als abgeschlossen, sobald mindestens eine der Ausgangsstellen taus von t im Unternetz markiert wird.

Ist eine komplexe Transition t aktiviert, kann diese schalten. Komplexe Transitionen schalten dabei in zwei Phasen. In der ersten Phase wird eine Instanz des Unternetzes AIN(t) erzeugt und die Markierungsfunktion m0(t) durchgeführt. Weiterhin werden die Markierungen aller Eingangsstellen der komplexen Transition (des übergeordneten AINs) entfernt. Die zweite Phase wird durch die Interaktionen im Unternetz beschrieben:

Sowohl das AIN als auch alle Instanzen von AIN(t) schalten unabhängig voneinander. Lediglich die Markierung der Ein- und Ausgangsstellen der komplexen Transitionen definieren die Schnittstellen. Dabei gelten folgende Regeln:

> *(1) Verändert eine Transition t∈ T eine Stelle s mit s∈ taus∪tein in einem AIN, so wird auch die entsprechende Stelle in allen Instanzen von AIN(t) verändert.*
>
> *(2) Wird durch das Schalten einer Transition t∈ T(t) eine Menge von Stellen St für die gilt, es existiert ein s∈ ST mit s∈taus oder s∈tein in einem Unternetz von AIN in*

ihrer Markierung verändert, so wird die Markierung von s auch im AIN und allen anderen Unternetzen von AIN verändert.

(3) Wird durch das Schalten einer Transition t∈ T(t) eine Menge von Stellen St verändert, und es existiert ein s∈ St mit s∉taus. So gilt die komplexe Interaktion, die durch t modelliert wurde als abgeschlossen. Die Instanz des Unternetzes AIN(t) wird dann vernichtet.

Diese Definition des Schaltverhaltens delegiert die Berechnung der neuen Markierung nach dem Schalten der komplexen Transition im AIN an das jeweilige Unternetz. Das Unternetz ist dabei für die Einhaltung der Schaltregeln verantwortlich.

Abbildung 22 veranschaulicht die Abhängigkeiten zwischen AIN und einem Unternetz AIN(T). Dargestellt werden drei Netze. Von links nach rechts sind dies: ein einfaches AIN, das Unternetz AIN(T) der Transition T und eine alternative Darstellung des Netzes AIN(T) genannt AIN', das Eigenschaften des Unternetzes in das AIN überträgt.

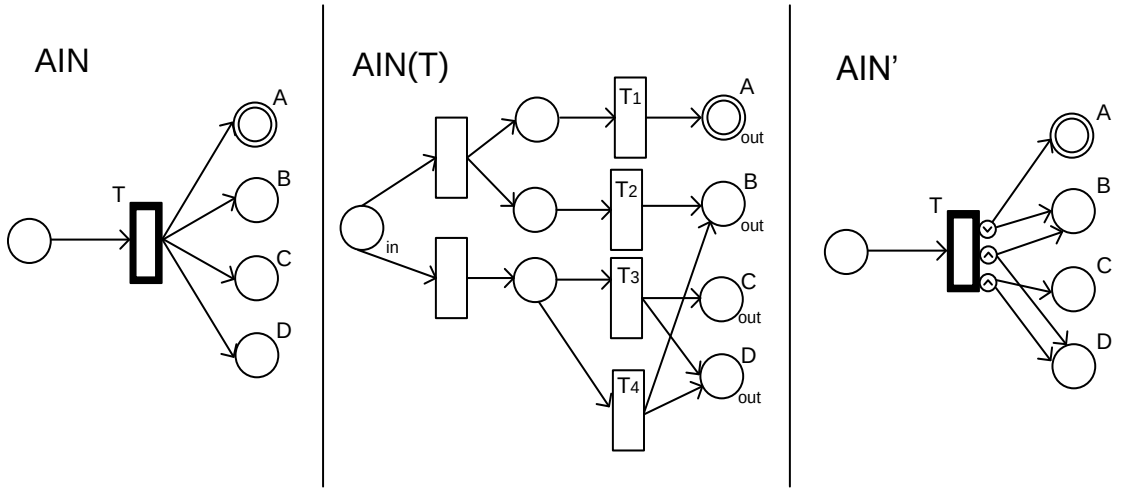
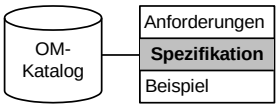


Abbildung 22: Komplexe Transitionen in einem AIN

Wie in dem Netz AIN visualisiert, werden komplexe Stellen durch eine schwarze Umrandung gekennzeichnet. Ausgangsstellen komplexer Transitionen werden im Unternetz mit einem „out" gekennzeichnet, damit ersichtlich ist, dass bei einer Manipulation der Markierung dieser Stellen die komplexe Transition schalten kann und sich damit beenden lässt. Eingangsstellen werden entsprechend mit einem „in" markiert. Im AIN in der Abbildung 22 besitzt die komplexe Transition T vier Ausgangsstellen (A, B, C, D), dabei können Ausgangsstellen unterschiedliche Kapazitäten besitzen (A hat die Kapazität unendlich, während B, C und D eine Kapazität von 1 haben). Das Unternetz AIN(T) verfügt, wie in der Definition gefordert, über alle vier Ausgangsstellen.



Laut Definition wird das Schalten von T beendet und damit das Unternetz AIN(T) zerstört, wenn mindestens eine der Ausgangsstellen von T im AIN(T) markiert wird. Betrachtet man die Möglichkeiten in AIN(T), die zu einem Beenden des Schaltens von T führen, so sind nicht alle Kombinationen der Menge {A, B, C, D} möglich, sondern nur vier:

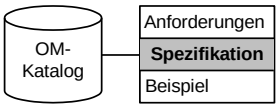
- {A} wenn T1 schaltet.
- {B} wenn T2 schaltet.
- {C,D} wenn T3 schaltet.
- {B,D} wenn T4 schaltet.

In AIN' wird eine Notation verwendet, in welcher das Wissen über mögliche Folgen des Schaltens komplexer Transitionen auch im übergeordneten Netz visualisiert wird. Die komplexe Transition wird dabei, wie in Abbildung 22 dargestellt, mit $\vee$ und $\wedge$ erweitert. Von der Transition ausgehende Kanten können nun an diesen Symbolen ansetzen, wobei nur ein $\vee$, aber beliebig viele $\wedge$ eine komplexe Transition erweitern können. Mit einem $\vee$ verbundene Stellen fassen alle Möglichkeiten zusammen, bei denen genau eine Ausgangsstelle im Unternetz markiert wird, wobei jedes $\wedge$ eine Variante beschreibt, bei der alle Stellen, die mit dem Symbol verknüpft sind, markiert werden. Diese Konstrukte wurden ausschließlich als graphisches Hilfsmittel für die Modellierung von AINs mit komplexen Transaktionen eingeführt und beeinflussen nicht die vorhergegangenen Definitionen.

Zusammenfassend muss angemerkt werden, dass das Konstrukt der komplexen Transitionen primär für die Kapselung von zusammengehörigen Modellteilen verwendet werden kann. Komplexe Transitionen können Logiken enthalten, die nur zu einem kleinen Teil aus dem übergeordneten Netz ersichtlich sind. Auch das außer Kraft treten der Schaltregeln für komplexe Transitionen trägt dazu bei. So muss ein Unternetz bekannt sein, wenn man Aussagen über das mögliche Verhalten, beziehungsweise über mögliche Interaktionen treffen möchte. Diese Entkopplung der Logik in Unternetzen kann aber auch bewusst eingesetzt werden, um noch nicht formalisierbare Sachverhalte zu modellieren, die erst im Laufe der Modellspezifikation konkretisiert werden können.

5.3.3 Modellierung von Systeminteraktionen

Interaktionen und deren Auswirkungen auf Objektmetaphern und damit auch auf die weiteren Interaktionsmöglichkeiten, sind nach der bisherigen Definition von AINs ausschließlich vom Benutzer abhängig. Vernachlässigt werden dabei Effekte aus anderen Quellen, die Interaktionsmöglichkeiten ändern können. Beispiele für solche Quellen sind dynamische Datenbestände wie Datenbanken oder auch externe Dienste, bei denen nicht vorhersehbar ist, ob der Benutzer auf eine Anfrage immer ein Ergebnis erhält. Bis zu einem gewissen Grad ist diese Vereinfachung auch haltbar. Ob das Ergebnis einer Katalogsuche nun ein Pullover oder ein Autoradio ist, ist erstmal für die weiteren Interaktionsmöglichkeiten und damit für das Interaktionsmodell nicht von Bedeutung. Betrachtet man aber beispielsweise das in Abbildung 21 modellierte Verhalten, so ist das Ergebnis einer Suche in einem Katalog notwendigerweise eine Produktliste, die



beispielsweise durch eine geordnete Menge von Verweisobjekten definiert ist. Dies mag aber nicht immer der Fall sein, da dies einerseits von der Suchanfrage des Benutzers abhängt, andererseits aber auch vom Datenbestand des Kataloges. Das Ergebnis der Suche kann auch nur ein Produkt sein, so dass eine Produktliste gar nicht mehr angezeigt werden muss. Auch auf der Ebene der Objektmetaphern sind die Resultate von Interaktionen nicht allein von den Interaktionen des Benutzers abhängig. Der Benutzer weiß lediglich, welche Interaktionen er mit diesen durchführen kann, er kann das Ergebnis aber nicht immer vorhersagen.

Betrachtet man die Arten von Interaktionen eingehender, bei denen externe Systeme Einfluss auf die Interaktionsmöglichkeiten haben (im Folgenden Systeminteraktionen genannt), so kann unterschieden werden zwischen:

- Interaktionen, die durch den Benutzer ausgelöst werden, deren Resultat aber von externen Gegebenheiten abhängt (im Folgenden auch benutzerinitiierte Systeminteraktionen genannt).
- Interaktionen, die selbst durch externe Ereignisse ausgelöst werden (im Folgenden externinitiierte Systeminteraktionen genannt), welche aber die Interaktionsmöglichkeiten des Benutzers beeinflussen.

Externinitiierten Systeminteraktionen liegen Informationssysteme zugrunde, die basierend auf Nutzerdaten aktiv werden, aber auch Dienste auslösen können. Diese werden nur noch indirekt, nämlich durch die Profileingabe, vom Benutzer ausgelöst. Die Informationslogistik (vgl. [DL01]) liefert eine Reihe von Beispielen für intelligente Informationssysteme, die auf Basis von Nutzerpräferenzen, Kontextinformationen und Bedarfsprofilen einen ganzen Pool an Dienst- und Informationsquellen überwachen und den Benutzer aktiv mit Informationen und Dienstleistungen versorgen.

Die Modellierung solcher Systeminteraktionen erfordert eine Erweiterung der Definition eines AIN.

> *Ein AIN_{ST} = (S, T, F, K, W, B, m0, Tk, Tbsi, Tesi) heißt AIN mit systeminitiierten Transitionen. Tbsi ist die Menge der benutzerinitiierten Systeminteraktionen und Tesi die Menge der externinitiierten Systeminteraktionen im Netz. Es gilt Tbsi $\cup$ Tesi $\subseteq$ Tk und Tbsi $\cap$ Tesi=ø.*

Formal werden Systeminteraktionen in AINs durch komplexe Transitionen modelliert. Die Unternetze von Systeminteraktionen sind dabei nicht mehr Teil des AINs und werden durch extern bereitgestellte Mechanismen realisiert. Dabei macht man sich die Eigenschaft von komplexen Interaktionen zu nutze, interne Abläufe vor dem übergeordneten Netz weitgehend zu verbergen. Das Schaltverhalten der Systeminteraktionen ist analog zu komplexen Transitionen definiert. Externinitiierte Systeminteraktionen werden nicht durch eine Nutzerinteraktion ausgelöst, sondern durch das Auftreten eines externen Ereignisses. Sowohl für benutzerinitiierte, als auch für systeminitiierte Interaktionen gilt, dass sie nur dann schalten können, wenn sie laut Definition aktiviert sind.

Obwohl keine Netze für die Beschreibung von systeminitiierten Interaktionen existieren, kann die eingeführte Erweiterung von komplexen Transitionen ($\wedge$ und $\vee$ Annotierung an den komplexen Transitionen) verwendet werden, um die möglichen Auswirkungen auf das AIN zu beschreiben. Es ist aber anzumerken, dass aus Sicht des AINs die neue Markierung der Ein- und Ausgangsstellen nach einem Schalten der Transition nicht deterministisch ist. Auch wenn keine Aussage getroffen werden kann, welcher der möglichen Fälle eintrifft, ist die Menge der Möglichkeiten beschrieben und endlich.

Ein Sonderfall im Rahmen der Systeminteraktionen ist die leere Objektmetapher. Die Suche in einem Katalog kann beispielsweise keine Resultate für bestimmte Suchparameter liefern. Zur Modellierung dieses Sachverhalts wird angenommen, dass jede Systeminteraktion über eine zusätzliche Kante zu seinen Eingangsstellen verfügt. Eine Modellierung dieser Kante ist nicht notwendig, da sich dadurch die Interaktionsmöglichkeiten des Nutzers nicht ändern, da die Eingangsstellen der Transition nach dem Schaltvorgang wieder markiert sind.

Abbildung 23 veranschaulicht die graphische Repräsentation von systeminitiierten Interaktionen. Graphisch werden Systeminteraktionen als eingefärbte Transitionen modelliert. Dabei werden benutzerinitiierte Systeminteraktionen durch eine partielle Einfärbung (rechte Hälfte der Transition) und externinitiierte Systeminteraktionen durch eine vollständige Einfärbung der Transition modelliert.

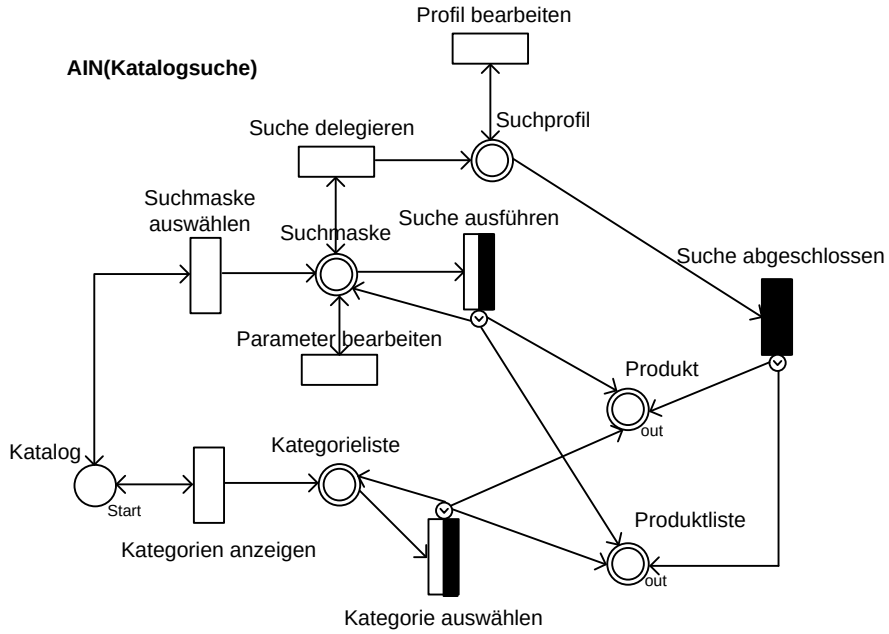
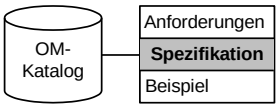


Abbildung 23: Systeminteraktionen eines AIN

Modelliert wird in Abbildung 23 ein Unternetz der komplexen Transition „Suchen im Katalog“, welches unterschiedliche Suchvorgänge in einem Katalog beschreibt. Die Suche kann dabei auf drei unterschiedliche Arten ausgeführt werden: einerseits durch die Navigation in Kategorien, durch eine Stichwortsuche



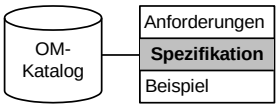
mit Hilfe von Suchmasken und schließlich durch einen speziellen Agenten, der eine Suche auf Basis des Suchprofils durchführt.

Wird die komplexe Interaktion „Suche im Katalog“ durch den Benutzer ausgeführt, so wird das in Abbildung 23 gezeigte AIN initiiert. Die Startmarkierungsfunktion m0 (Katalogsuche) ist dabei leer. Die Objektmetapher Katalog bietet dem Benutzer die Suche über eine Suchmaske oder über Kategorien an. Wird eine Suche über die Kategorien gewählt, so erhält der Benutzer Zugriff auf eine Kategorieliste. Die Kategorieliste bietet lediglich die benutzerinitiierte Interaktion „Kategorie auswählen“ an. Der Benutzer initiiert die Interaktion durch die Auswahl einer Kategorie. Welches Ergebnis diese Interaktion zur Folge hat, ist abhängig von den Kategorien und den Inhalten, die den Kategorien zugeordnet wurden. So wird eine Kategorieliste zurückgeliefert, wenn die ausgewählte Kategorie über Unterkategorien verfügt, oder eine Produktliste, falls dieser Kategorie ein Blatt im Kategoriebaum zugeordnet wurde, dem nur noch Produkte zugeordnet sind. Enthält die ausgewählte Kategorie nur ein Produkt, so wird dieses zurückgeliefert. Das Symbol $\vee$ zeigt dabei an, dass nur eine der drei Alternativen eintreten wird. Die in diesem Abschnitt beschriebene Logik (z.B. wenn die ausgewählte Kategorie Unterkategorien besitzt, wird die Stelle Kategorieliste markiert) findet keine Berücksichtigung im Netz. Die Transition „Kategorie auswählen“ könnte auch eine völlig andere Logik realisieren. Wenn beispielsweise die ausgewählte Kategorie Unterkategorien besitzt, wird die Stelle Produktliste markiert, die alle Produkte dieser und aller Unterkategorien enthält. Dennoch bildet das Netz die Interaktionsmöglichkeiten korrekt ab und erfüllt somit den Anspruch, eine abstrakte Beschreibung zu erlauben.

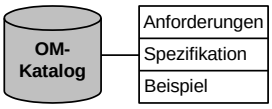
Ein Beispiel für eine externinitiierte Interaktion ist die Transition „Suche abgeschlossen“. Diese Transition kann nur dann schalten, wenn Suchprofile existieren. Ein Suchprofil kann durch eine Interaktion des Benutzers aus einer Suchmaske generiert und jederzeit vom Benutzer modifiziert werden. Existierende Suchprofile werden über Markierungen modelliert. Sobald eine Markierung modifiziert wird, hat dies im System zur Folge, dass die Transition „Suche abgeschlossen“ nur dann schalten kann, wenn ein Suchergebnis für die veränderte Markierung gefunden wurde. Problematisch an dieser Form der Modellierung ist die Synchronisation von benutzer- und systeminitiierten Interaktionen. So ist es möglich, dass während der Benutzer gerade sein Suchprofil bearbeitet, zeitgleich durch eine externinitiierte Interaktionen eine weitere Transition ausgelöst wird. AINs bieten keine Methoden, um diese Nebenläufigkeiten aufzulösen, es sollte aber dem Modellierer bewusst sein, dass solche Probleme auftreten können.

Wann und mit welchem Ergebnis eine externinitiierte Transition schaltet, wird von außen bestimmt. In dem Beispiel nimmt ein Agent die Rolle des Systems an, das genau dann schaltet, wenn ein Ergebnis vorliegt. Im AIN werden lediglich die Möglichkeiten modelliert, welches Ereignis eintrifft.

In Dialognetzen gibt es keine Analogie zu Systeminteraktionen. Hier ist die Modellierung der Fensterabfolgen einzig und allein von Nutzerinteraktionen abhängig. Lediglich die Interaktion innerhalb eines Fensters wird mit Hilfe von Constraints und Regelsystemen dargestellt, die auf Basis vordefinierter Ereignisse und Bedingungen bestimmte Aktionen innerhalb eines Fensters aufrufen können. Bedingungen



und Ereignisse können dabei auch extern ausgelöst werden oder Daten beinhalten, die aus externen Systemen bezogen werden. Es ist allerdings aus dem Modell ersichtlich, welche Auswirkungen bei einer bestimmten Daten- und Ereignismenge zu erwarten sind, während AIN lediglich modellieren, dass Möglichkeiten existieren. Welche und zu welchen Bedingungen diese eintreffen, wird außer acht gelassen. Dialognetze sind im Vergleich zu AINs systemnah, da sie auch nicht interaktionsbeschreibende Funktionen mit in die Modellierung integrieren. Im Gegensatz dazu ermöglichen es AINs komplexes Systemverhalten völlig transparent für das Modell zu gestalten. Diese Eigenschaft ist von großer Bedeutung für Systeme, die mit Hilfe externer Sensorik und/oder komplexen Semantiken arbeiten, bei denen das Systemverhalten einfach zu beschreiben ist.



5.4 Modellierung mit Hilfe des Objektmetaphern-Kataloges

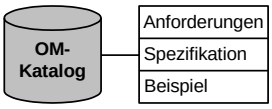
Die in den vorhergehenden Abschnitten eingeführten AINs beschreiben eine Methodik zur abstrakten Modellierung von Benutzerschnittstellen. Um diese Modellierung im Rahmen des „Compositional Modeling“ nutzbar zu machen, müssen Regelungen für den Modellierungsprozess definiert werden. Dem Baukasten-orientierten Ansatz entsprechend muss sich auch die Interaktionsmodellierung möglichst existierender Bausteine bedienen. Für die Organisation solcher Bausteine für die Interaktionsmodellierung wird der Objektmetaphern-Katalog (im folgenden OM-Katalog genannt) eingeführt.

Der OM-Katalog ist das primäre Werkzeug zur Modellierung der Interaktionsbeschreibung, welcher alle Elemente des AINs (also alle Objekt- und Interaktionsmetaphern) umfasst, mit denen die Interaktionsbeschreibung definiert wird. Durch diese Restriktion wird einerseits sichergestellt, dass Elemente des OM-Kataloges wieder verwendet werden, andererseits wird der OM-Katalog auch benutzt, um Zusatzinformationen über die modellierten Elemente ablegen zu können. Schließlich ermöglicht es der OM-Katalog auch Beziehungen zwischen der Interaktionsbeschreibung und den realen Bausteinen herzustellen, welche die Benutzerschnittstelle realisieren. Damit wird der Ansatz der Wiederverwendung bereits in der Modellierungsphase verankert.

Organisiert ist der OM-Katalog nach Objektmetaphern. Diese beschreiben domänenspezifische Entitäten, die auch den primären Ansatz zur Erstellung der Interaktionsbeschreibung liefern (vgl. Abschnitt 5.2). Für die Modellierung der Interaktionsbeschreibung werden die Objektmetaphern des OM-Kataloges analog zu Klassen in objektorientierten Systemen (vgl. [For02]) verwendet. Objektmetaphern des OM-Kataloges sind dementsprechend Vorlagen für die Modellierung der Interaktionsbeschreibung. In einem OM-Katalog werden Objektmetaphern einer Anwendungsdomäne (bspw. „Adipositas-Begleiter“, „Kaufhaus“) katalogisiert und dem Benutzer für die Modellierung zur Verfügung gestellt.

Formal beschrieben wird eine Objektmetapher durch eine Menge von Interaktionsmetaphern, die mit oder auf dieser Objektmetapher möglich sind.

Abbildung 24 veranschaulicht die Struktur eines OM-Katalogs mit Hilfe von Beschreibungselementen eines UML Klassen-Diagramms (vgl. [Bal05]).



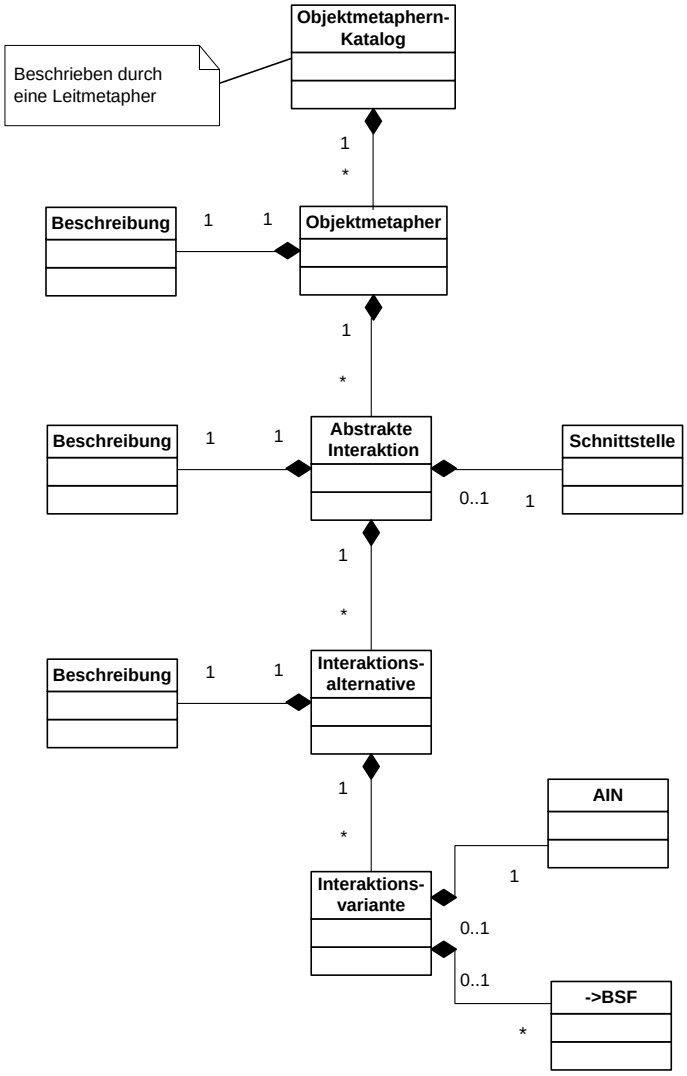
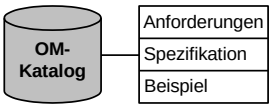


Abbildung 24: Struktur des Objektmetaphern-Kataloges

Die Grundlage für die Katalogisierung von Objektmetaphern ist die Definition einer „Leitmetapher“, welche die Anwendungsdomäne beschreibt. Diese Metapher gibt den verwendeten Objektmetaphern einen Rahmen und stellt eine „conceptual domain“ dar, wie sie in Abschnitt 5.2.3.2 definiert wurde. Damit verbunden ist auch die Definition der „target domain“, also der Klasse von Anwendungen, die mit Hilfe des Kataloges beschrieben werden können. Dabei ist zu beachten, dass Objektmetaphern unterschiedlich definiert sein können, wenn sie in unterschiedlichen Anwendungsdomänen (und damit auch unterschiedlichen OM-Katalogen) verwendet werden.

Für eine Anwendungsdomäne werden Objektmetaphern katalogisiert und über den OM-Katalog dem Modellierer zur Verfügung gestellt. Wie die Bezeichnung „OM-Katalog“ impliziert, werden Objektmetaphern zur Strukturierung des Kataloges herangezogen. Objektmetaphern wurde hier der Vorzug vor Interaktionsmetaphern gegeben, um dem Modellierer einen intuitiven Zugang zur Modellierung zu ermöglichen. Interaktionsmöglichkeiten werden durch verfügbare Objektmetaphern beschrieben.



Es existiert somit eine natürliche Assoziation von Objekten zu den möglichen Interaktionen, die durch den Katalog unterstützt wird.

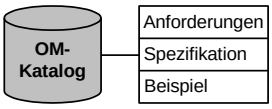
Beschrieben werden Objektmetaphern primär durch die Interaktionen, die mit oder auf diesen durchgeführt werden können. Wie in Abbildung 24 dargestellt, nutzt der OM-Katalog drei Konstrukte zur Organisation von Interaktionsmetaphern:

- **Abstrakte Interaktionen** fassen Interaktionen zusammen, die aus Sicht des Benutzers zur gleichen Zielverfolgung benutzt werden. Abstrakte Interaktionen ermöglichen somit eine Klassifizierung von Interaktionen.
  Weiterhin definieren sie die Schnittstellen einer Interaktion. Beschrieben wird die Schnittstelle durch Eingangs- und Ausgangsstellen (der Transition in einem AIN), also eingehende und ausgehende Objektmetaphern. Diese Objektmetaphern müssen ebenfalls im OM-Katalog definiert sein.
- Abstrakte Interaktionen können nun 1 bis n **Alternativen** anbieten. Alternativen sind Interaktionsansätze, die auf unterschiedlichen Methoden beruhen, aber zum selben Ergebnis führen. So ist beispielsweise die Navigation über Kategorien eine Alternative zur Suche über eine Suchmaske, wobei beide unter derselben Abstrakten Interaktion „Suche“ zusammengefasst werden können.
- **Varianten** definieren schließlich Unterschiede in diesen Alternativen und beschreiben die Interaktionsdurchführung. Beschrieben wird die Interaktion entweder durch ein AIN oder durch ein Benutzerschnittstellenfragment (in Abbildung 24 mit BSF abgekürzt). Benutzerschnittstellenfragmente werden in Abschnitt 7 definiert und beschreiben die wieder verwendbaren und ausführbaren Teile einer Benutzerschnittstelle.

Zur Modellierung der Interaktionsbeschreibung müssen nicht notwendigerweise Varianten verwendet werden. Möchte der Modellierer, dass erst zur Generierungszeit und somit unter Einbezug des Nutzungskontextes eine Variante ausgewählt wird, so können auch „Abstrakte Interaktionen“ oder Alternativen verwendet werden.

Varianten beinhalten schließlich die Beschreibung der Interaktionen. Beschrieben wird eine Interaktion rekursiv, indem eine Interaktion selbst durch ein AIN modelliert wird. Eine Variante wird dabei entweder durch ein AIN beschrieben, oder – die Endbedingung der Rekursion – indem ein Benutzerschnittstellenfragment[28] (kurz BSF genannt) der „Domain Theory“ (vgl. Abschnitt 7.3) referenziert wird, das diese Interaktion realisiert. Das AIN, das selbst zur Beschreibung einer Interaktion benutzt wird, verwendet dabei ebenfalls Objekt- und Interaktionsmetaphern, die im OM-Katalog definiert sind. Weiterhin besteht die Möglichkeit, dass sowohl ein AIN, als auch eine Referenz zu einem BSF angegeben ist. In diesem

[28] Benutzerschnittstellenfragmente beschreiben implementierungsnahe Bausteine, aus denen ausführbare Benutzerschnittstellen generiert werden können. Eine Definition und Spezifikation erfolgt in Kapitel 7 dieser Arbeit.



Fall wird zur Generierungszeit durch die Auswertung des Nutzungskontexts entschieden, ob das referenzierte BSF einzusetzen ist, oder ob die im AIN beschriebene Realisierung zu bevorzugen ist.

5.5 Instanzierung eines Objektmetaphern-Kataloges am Beispiel des Adipositas-Begleiters

Nachdem im vorhergehenden Abschnitt 5.4 definiert wurde, wie ein OM-Katalog aufgebaut ist, wird nun anhand des Beispiels „Adipositas-Begleiter“ erläutert, wie solch ein Katalog im realen Einsatz konzipiert und entwickelt werden kann. Im Zuge der Konzeption wird insbesondere dargestellt, wie das „Metaphern-Konzept“ der Interaktionsbeschreibung wichtige Fundamente für den Entwurf schafft.

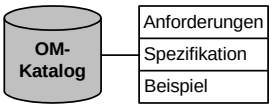
In der Gelderlandklinik lernen die Patienten in einer 6- bis 8-wöchigen stationären Therapie sich gesünder zu ernähren, sich richtig zu bewegen und schließlich emotional kontrollierter zu leben (Vermeidung von Fressattacken). In diesem Zeitraum werden ihnen Inhalte vermittelt, in Gruppen- und Einzelsitzungen Übungen durchgeführt und Hilfsmittel zur Hand gegeben wie beispielsweise Tagebücher, Pläne und Merkblätter. Obwohl die Patienten auch Einzeltherapien durchlaufen, verfügen sie nach der stationären Therapie über einen großen gemeinsamen Erfahrungsschatz.

Diese Erfahrungen können dazu benutzt werden, eine „source-domain“ zu beschreiben, wie sie in Abschnitt 5.4 für den Aufbau eines Objektmetaphern-Kataloges gefordert wird. Die „source domain“ bedient sich dabei real erlebter Erfahrungen, um damit die Interaktion mit einem IT-System zu beschreiben.

Die abstrakte Beschreibung der Benutzerschnittstellen mit Hilfe von Objektmetaphern wird daher Objekte und Assoziationen nutzen, die in der stationären Therapie erlernt wurden. Als Beispiel wird an dieser Stelle ein durch den Ernährungsplan definierter Ausschnitt aus dem Bereich „Ernährung“ vorgestellt (dargestellt in Anlehnung an [KLWK06]). Diese Beschreibung soll zu einem besseren Verständnis der verwendeten Objekt- und Interaktionsmetaphern führen, die in den folgenden Abschnitten die Grundlage für die Darstellung des Generierungsprozesses liefern werden.

Im Bereich Ernährung wurde als zugrunde liegendes Bewertungsschema ein lebensmittelorientiertes (nicht energieorientiertes) *Ampelsystem* gewählt, das Nahrungsmittel in drei Klassen (rot, gelb und grün) unterteilt. Nahrungsmittel, die „grün“ klassifiziert wurden, sind als gesund eingestuft, „gelbe“ Nahrungsmittel sind in Maßen genossen unproblematisch, während „rote“ Nahrungsmittel möglichst zu meiden sind. Ein weiteres Merkmal des Ampelsystems ist es, dem Nutzer Alternativangebote einer günstigeren Nahrungsmittelklasse anzubieten, die eine gesündere Alternative darstellen.

Dabei sind „rote“ Nahrungsmittel nicht verboten, sondern sollten in einem vernünftigen Verhältnis zu den anderen Klassen stehen. Das Definieren und Kontrollieren dieses Verhältnisses ist eine der wichtigsten Zielsetzungen der Therapie. Zunächst legt der Patient zusammen mit seinem Ernährungstherapeuten fest, in welchem Verhältnis rote, gelbe oder grüne Nahrungsmittel auf seinem Speiseplan stehen dürfen. Essenszeiten für die Hauptmahlzeiten (Frühstück, Mittag- und Abendessen) werden wöchentlich geplant und festgelegt. Bis zu zwei Zwischenmahlzeiten werden vom Patienten ungeplant eingenommen.

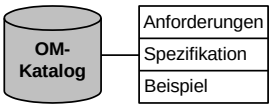


Alle Mahlzeiten (auch die Zwischenmahlzeiten) werden in einem Esstagebuch protokolliert. Protokolliert wird dabei nicht nur, wann und was gegessen wurde, sondern auch die emotionale Situation des Patienten nach dem Essen und das Sättigungsgefühl. Zielsetzung ist es dabei, einerseits zu einer objektiven Einschätzung des eigenen Essverhaltens zu gelangen, andererseits durch regelmäßige Nahrungsaufnahme zu starke Hungergefühle zu vermeiden, die zu „Essattacken“ führen können.

Den Patienten steht ein persönliches Rezeptbuch zur Verfügung, das vorgegebene Rezepte enthält, aber auch die Möglichkeit bietet, eigene Gerichte zusammenstellen. Bei der Planung der Gerichte nutzen die Patienten ebenfalls das Ampelsystem, und können selbstständig mögliche Alternativen zu den Zutaten aufführen.

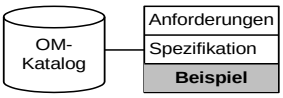
Tabelle 2 beschreibt Objektmetaphern aus dem Bereich Ernährung anhand der in Abschnitt 5.4 eingeführten Struktur des Objektmetaphern-Kataloges. Um die Übersichtlichkeit zu wahren, sind lediglich „Objektmetaphern“ und „Abstrakte Interaktionen“ aufgeführt. Weiterhin wurde eine Spalte „Beschreibung der Objektmetaphern“ eingeführt, welche die Assoziationen der Patienten mit der Objektmetapher darlegt. Auf eine Beschreibung der „Alternativen“ und „Varianten“ wurde daher verzichtet.

Leitmetapher	Objektmetapher	Abstrakte Interaktionen	Beschreibung der Objektmetapher
Stationäre Therapie in der Gelderland Klinik	Ernährungsplan	Planen	Eine der Kernmetaphern für die Ernährung. Der Ernährungsplan ist dabei zentrales Werkzeug für die Zieldefinition, die Planung, die Protokollierung und auch die Analyse.
		Bewerten	
		Analysieren	
		Zielerfüllung betrachten	
		Hilfen annehmen	
	Rezeptbuch	Rezept suchen	Ein beispielsweise durch einen Schnellhefter realisiertes Buch. Kategorisiert ist es nach Gerichten der drei Hauptmahlzeiten und der Zwischenmahlzeiten.
		Rezept hinzufügen	
		Rezept ändern	
		Rezept löschen	
	Rezept	Rezept ändern	Ein klassisches Rezept wie es in jedem Kochbuch zu finden ist. Erweitert ist es um das Ampelsystem.
		Rezept hinzufügen	
		Rezept löschen	
		Ampel setzen	
	Lebensmittel	Ampel setzen	Eine klassische Zutat in einem Rezept, die um das Ampelsystem erweitert wurde.
	Ernährungsziel	Ziel festsetzen	Ernährungsziele sind prozentua-



		Zeitraum festlegen	le Angaben, wie viele grüne, gelbe und rote Lebensmittel an einem Tag gegessen wurden. Definiert wird dabei lediglich das Verhältnis zwischen den drei Nahrungsmittelklassen.
	Ernährungsampel	setzen	Repräsentiert das Ampelsystem.
	Bewertung	Rezept bewerten	Repräsentiert ein Bewertungsschema, das in der Klinik eingesetzt wird.
		Befinden bewerten	
		Sättigungsgefühl bewerten	
	Einkaufsliste	Liste ergänzen	Eine klassische Einkaufsliste bestehend aus einzelnen Einträgen, die abgearbeitet werden müssen.
		Eintrag abhaken	
	Mahlzeit	Mahlzeit bewerten	Eine Mahlzeit ist entweder eine Haupt- oder Zwischenmahlzeit und ist immer terminiert.
		Rezept ändern	
		Uhrzeit ändern	
	Wochenplan	Zielerfüllung betrachten	Ist wie ein „Stundenplan“ aufgeteilt, wobei für jeden Tag die Möglichkeit zur Definition der drei Hauptmahlzeiten und zwei Zwischenmahlzeiten besteht.
		Einkaufsliste anzeigen	
	Tagesplan	Mahlzeit auswählen	Beschreibt die Mahlzeiten eines Tages.
		Mahlzeit festlegen	
	Uhr	Urzeit festlegen	Repräsentiert die Möglichkeit eine Uhrzeit zu festzulegen.
		Urzeit ändern	

Tabelle 2: Ausschnitt aus dem OM-Katalog für den Adipositas-Begleiter



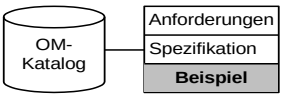
5.6 Interaktionsbeschreibung am Beispiel des Adipositas-Begleiters

Im Rahmen des Generierungsprozesses stellt die Interaktionsbeschreibung die zentrale Informationsquelle für die Inhalte der zu generierenden Benutzerschnittstelle dar. Sie liefert Informationen über mögliche Interaktionen oder mögliche Interaktionsabläufe und beantwortet die Frage, wie die Benutzerschnittstelle auf Nutzereingaben reagiert. Wesentliches Merkmal dieser Beschreibung ist, dass sie frei von geräte- und anwenderspezifischen Eigenschaften ist und somit die Ausgangslage für die Generierung unterschiedlicher Realisierungen sein kann.

Als formale Beschreibungsmethodik wurden Abstract Interaction Nets, wie sie in Abschnitt 5.3 eingeführt wurden, verwendet. Entsprechend dem Entwicklungsprozess (vgl. Abschnitt 4.3) erfolgt die Modellierung der Interaktionsbeschreibung in Verbindung mit einem Objektmetaphern-Katalog (vgl. Abschnitt 5.4). Der Objektmetaphern-Katalog stellt einerseits Bausteine (Objekt- und Interaktionsmetapher) für die Interaktionsmodellierung einer Anwendungsdomäne bereit und ermöglicht andererseits den Aufbau einer Wissensbasis, indem dort zusätzliche Informationen zu den modellierten Elementen abgelegt werden. Die Modellierung der Interaktionsbeschreibung beinhaltet somit nicht nur die Erstellung eines AIN, sondern erfordert im gleichen Maße die Beschreibung der Modellierungselemente im Objektmetaphern-Katalog (vgl. Abschnitt 5.4).

Erfahrungen in der Modellierung mit AINs haben gezeigt, dass ein Top-Down-Ansatz für die Erstellung der Interaktionsbeschreibung besonders geeignet ist. Analog zu Task-basierten Spezifikationsmethoden (vgl. Abschnitt 5.1.4.2) erfolgt in einem ersten Schritt die Definition von Aufgaben bzw. Zielen, die durch Dekomposition zunehmend detailliert werden. AINs unterstützen dieses Vorgehen durch die Bereitstellung komplexer Transitionen (vgl. Abschnitt 5.3.2). Eine komplexe Transition fasst ganze Interaktionsabläufe zusammen, und wird selbst wieder durch ein AIN beschrieben. Unterstützt wird dieses Vorgehen zusätzlich durch die Struktur des Objektmetaphern-Kataloges. Hier erfolgt die Beschreibung von Interaktionsmetaphern ebenfalls durch ein AIN (und/oder den Verweis auf Benutzerschnittstellenfragmente), so dass Interaktionsmetaphern selbst wieder als komplexe Transitionen aufgefasst werden können.

Überträgt man diesen Ansatz auf die Beschreibung des Adipositas-Begleiters, wie er in Abschnitt 3.3 eingeführt wurde, so kann der Adipositas-Begleiter selbst als Objektmetapher aufgefasst werden, mit der die Patienten interagieren können.



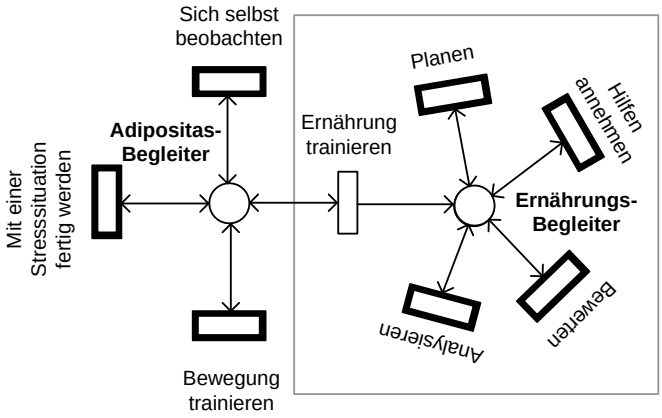


Abbildung 25: Strukturierung des Adipositas-Begleiters mit Hilfe von AINs

Abbildung 25 visualisiert ein mögliches Interaktionsmodell des Adipositas-Begleiters mit Hilfe eines AINs und veranschaulicht, wie durch Dekomposition der Interaktionsmetaphern schrittweise eine Spezifikation entstehen kann. Die Stelle „Adipositas-Begleiter“ in Abbildung 25 repräsentiert die zentrale Objektmetapher, und die komplexen Transitionen die Interaktionsmetaphern („Bewegung trainieren“, „Mit einer Stresssituation fertig werden“, „Ernährung trainieren“ usw.), die der Nutzer mit dieser Objektmetapher durchführen kann. Die Interaktionsmetaphern spiegeln dabei die zentralen Bereiche der Adipositas-Therapie wieder, indem sie Therapieaktivitäten für die Beschreibung nutzen. Durch die Verwendung von komplexen Transitionen ist es möglich, ganze Anwendungsbereiche zusammen zu fassen. Das Konstrukt der Nebenstelle[29] (Adipositas-Begleiter) erlaubt es weiterhin abzubilden, dass Nutzer jederzeit (auch gleichzeitig) Zugriff auf die vier Therapiebereiche haben und frei entscheiden können, welchen dieser Bereiche sie nutzen wollen.

Rechts in Abbildung 25 wird die Dekomposition der komplexen Transition „Ernährung trainieren“ dargestellt. Dadurch wird der Therapiebereich „Ernährung“ modelliert. Dies entspricht auch weitgehend den Erfahrungen der Patienten in der Klinik, wo sich die einzelnen Therapiebereiche durch unterschiedliche Therapeuten und Therapieangebote auszeichnen. Der „Ernährungs-Begleiter“ soll als Objektmetapher diese Struktur abbilden. Die Interaktionsmetaphern, die der Ernährungs-Begleiter anbietet, spezifizieren nun genauer die möglichen Aktivitäten des Patienten, die natürlich weitergehend verfeinert werden müssen.

Eine abstrakte Darstellung ist ein hilfreiches Mittel für eine erste Klassifizierung komplexer Benutzerschnittstellen und bietet bereits die Möglichkeit, wichtige Sachverhalte abzubilden. So wird beispielsweise modelliert, dass Patienten möglichst durchgehend Zugriff auf alle Therapiebereiche haben sollen. Dies wird durch die Verwendung von Nebenstellen modelliert, so dass die Objektmetapher „Adipositas-

[29] Nebenstellen sind Stellen im AIN, die für eine Transition gleichzeitig Ein- und Ausgangsstelle sind (vgl. Abschnitt 5.3.1).

Begleiter“ bei der Durchführung jeder der Interaktionen verfügbar bleibt. Andererseits soll das Beispiel aber auch verdeutlichen, dass eine Interaktionsmodellierung auf dem Abstraktionsniveau nicht ausreichend sein kann, um daraus Benutzerschnittstellen generieren zu können.

Anhand eines konkreteren Beispiels aus dem Bereich Ernährung wird eine Interaktionsbeschreibung vorgestellt, welche im späteren Verlauf (vgl. Kapitel 8) die Ausgangslage für einen Generierungsprozess sein wird. Ausgewählt wurde hierzu die Interaktionsbeschreibung für die Planung einer Mahlzeit.

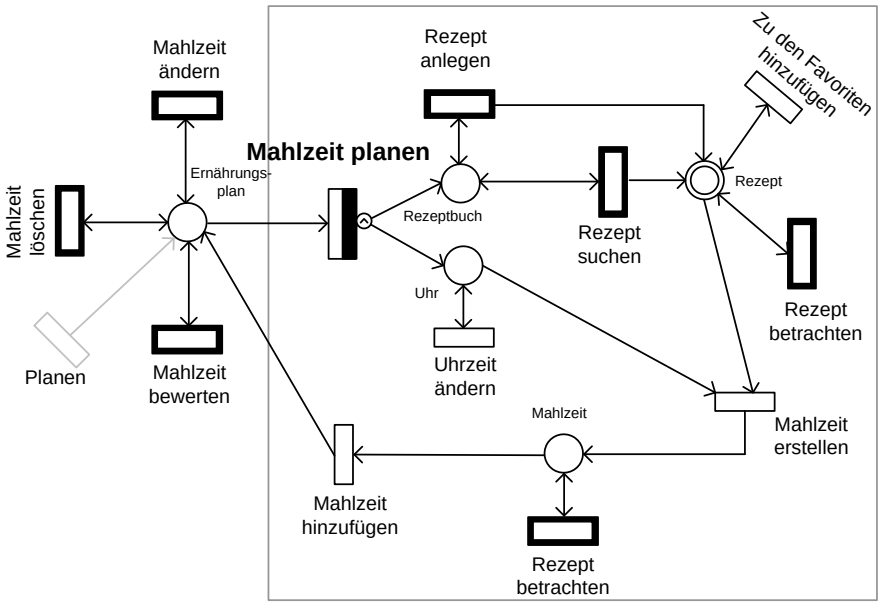
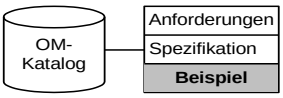


Abbildung 26: AIN für die komplexe Interaktion „Mahlzeit planen“

Abbildung 26 visualisiert eine Interaktionsbeschreibung für die Interaktion Ernährungsplanung, die beispielsweise von der Interaktionsmetapher „Planen“ aus Abbildung 25 gestartet wird. Die Transition „Planen“ wurde in Abbildung 26 hellgrau dargestellt, um anzuzeigen, dass dies die Interaktion ist, welche den Ernährungsplan aktiviert.

Zentrale Objektmetapher der Ernähungsplanung ist der „Ernährungsplan“. Patienten der Gelderland-Klinik verbinden mit dem Ernährungsplan eine Art Stundenplan (wöchentliche Planung), in dem sie für jeden Tag drei Hauptmahlzeiten (Frühstück, Mittag- und Abendessen) und zwei Zwischenmahlzeiten definieren können. Geplant werden die Hauptmahlzeiten, indem festgelegt wird, welches Gericht gekocht wird und weiterhin die Uhrzeit, wann die jeweilige Mahlzeit erfolgen soll (siehe Abschnitt 3.3).

Die Transition „Mahlzeit planen“ wurde als Nebenstelle realisiert, um darzustellen, dass diese die gleichzeitige Durchführung anderer Interaktionen („Mahlzeit löschen“, „Mahlzeit ändern“ usw.) erlaubt. Dabei beschreiben die Interaktionen „Mahlzeit bewerten“, „Mahlzeit ändern“ und „Mahlzeit planen“ Interaktionen, die über die gleiche Objektmetapher ausgelöst werden können. Die komplexe Transition „Mahlzeit planen“ wird durch den rechteckigen Kasten rechts in Abbildung 26 spezifiziert.



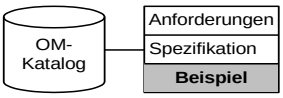
Anzumerken ist, dass diese Transition als „benutzerinitiierte Systeminteraktion“ (vgl. Abschnitt 5.3.3) gekennzeichnet ist, und damit modelliert, dass ein Benutzer zwar diese Interaktion auslöst, das Ergebnis dieser Transition aber von einem nicht im AIN modellierten Datensatz abhängig ist. Hier wird die Systeminteraktion benutzt um zu modellieren, dass eine Planung nicht möglich ist, wenn bereits eine Mahlzeit geplant wurde. In diesem Fall wird eine andere Reaktion der Benutzerschnittstelle erwartet, die aber nicht mehr Teil der betrachteten Interaktionsbeschreibung ist, was in Abbildung 26 durch die „$\wedge$ Erweiterung“ angedeutet wird.

Wird die Interaktion „Mahlzeit planen“ ausgeführt, so werden die Stellen Rezeptbuch und Uhr markiert und damit beide Objektmetaphern verfügbar. Das Rezeptbuch bietet zwei komplexe Interaktionen, die es dem Nutzer ermöglichen, ein Rezept auszuwählen. Es besteht die Möglichkeit, in einem Rezeptbuch zu suchen oder selbst ein neues Rezept zu definieren. Das Ergebnis beider Alternativen ist ein Rezept, das betrachtet oder aber zu einer „Favoriten-Liste“ hinzugefügt werden kann. Die „Favoriten-Liste“ bietet einen Schnellzugriff auf oftmals benutzte Rezepte und wird im „realen Rezeptbuch“ durch Post-its realisiert. Die Stelle Rezept ist dabei als Stelle mit unbeschränkter Kapazität (umrandeter Kreis) gekennzeichnet, womit modelliert wird, dass Nutzer auch mehrere Rezepte suchen bzw. erzeugen können. Die Transition Mahlzeit erstellen wird verfügbar, sobald eine Uhrzeit und mindestens ein Rezept verfügbar sind. Diese Interaktion ermöglicht schließlich die Erzeugung einer Mahlzeit.

Sobald eine Mahlzeit verfügbar ist, kann diese vom Benutzer betrachtet und zu dem Ernährungsplan hinzugefügt werden. Mit dem Hinzufügen der Mahlzeit zum Ernährungsplan gilt die komplexe Interaktion „Mahlzeit planen“ als abgeschlossen. Alle Markierungen in den Stellen (auch in der komplexen Interaktion) werden aufgelöst.

Die Granularität der Beschreibung ist unter anderem auch von dem verfügbaren Objektmetaphern-Katalog abhängig. Dem Modellierer steht es dabei frei, dem Katalog neue Objekt- und Interaktionsmetaphern hinzuzufügen oder existierende Objektmetaphern durch neue Interaktionsmetaphern zu erweitern. Eine Interaktionsbeschreibung gilt erst dann als vollständig und somit geeignet, um daraus einen Generierungsprozess anzustoßen, wenn jede verwendete Objekt- und Interaktionsmetapher mit ihren notwendigen Attributen im Objektmetaphern-Katalog aufgenommen wurden. Da das für die Interaktionsbeschreibung verwendete AIN weitgehend frei von Zusatzinformationen (bspw. Constraints, Funktionsschnittstellen, Konfigurationen usw.) ist, wird hierdurch sichergestellt, dass alle für die Generierung notwendigen Informationen auch tatsächlich vorhanden sind. Weiterhin erfordert die Definition neuer Interaktionsmetaphern auch eine Beschreibung durch ein AIN und/oder die Zuordnung eines BSF (vgl. Abschnitt 5.4). Diese Beschreibung ist dabei nicht mit einer komplexen Transition in einem AIN zu verwechseln. Die Strukturierung der Interaktionsmetaphern (Alternativen, Varianten) beschreibt unterschiedliche Möglichkeiten für die Realisierung der Interaktion, deren Auswahl aber vom Nutzungskontext abhängig ist.

Auf eine genauere Beschreibung des verwendeten Objektmetaphern-Kataloges wird an dieser Stelle verzichtet und auf den folgenden Abschnitt verwiesen. Dort wird dargestellt, wie die Informationen des



Objektmetaphern-Kataloges benutzt werden, um BSF[30] für die Realisierung der Interaktionsbeschreibung zu ermitteln.

5.7 Zusammenfassung und abschließende Betrachtung

Das Ziel dieses Kapitels war die Bereitstellung einer Modellierungsmethodik für eine Interaktionsbeschreibung, wie sie im Rahmen des „Compositional Modeling“ benötigt wird, um daraus ausführbare Benutzerschnittstellen zu generieren.

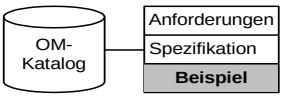
Die Untersuchung bereits existierender Modellierungsansätze (vgl. Abschnitt 5.1.4) hat gezeigt, dass dafür eine gewisse Balance zwischen Abstraktionslevel und Nähe zur Benutzerschnittstelle notwendig ist, die spezielle Anforderungen an die Modellierungsmethodik stellt. Objektmetaphern wurden basierend auf diesen Anforderungen als favorisierter Lösungsansatz gewählt, und mit AINs eine entsprechende Spezifikation definiert.

Die Nutzung von AINs als Interaktionsbeschreibung setzt aber weitergehendes Wissen über die semantische Bedeutung der modellierten Stellen und Transitionen voraus.

Zieht man auch hier eine Analogie zu Dialognetzen, so beschreiben Dialognetze mit ihren Modellierungselementen fensterbasierte Systeme: Transitionen sind Interaktionsobjekte (Buttons, Verweise, Auswahlboxen), und Stellen repräsentieren genau ein Fenster. Auch wenn AINs für die Modellierung nur zwei Modellelemente (Stellen und Transitionen) nutzen, sind die Objektmetaphern völlig unterschiedlich in ihrer Bedeutung. So kann die Transition „Rezept suchen“ in Abbildung 26 beispielsweise durch eine Navigation in Kategorien, durch Suchmaske oder sogar durch ein Agentensystem realisiert werden, das basierend auf Nutzerpräferenzen und dem aktuellen Informationsbedarf asynchron für den Benutzer aktiv ist. Stellen, Transitionen und Relationen können nicht ausreichend sein, um diese Informationen zu kodieren. Die Nutzung von AINs für die Generierung von Benutzerschnittstellen erfordert weitere Informationen über die Bedeutung der verwendeten Stellen und Transitionen. Hierfür liefert der gewählte Metaphernansatz wiederum eine interessante Lösung.

Es liegt nahe, Objekt- und Interaktionsmetaphern einer Domäne zu katalogisieren, und dem Interaktionsmodellierer einen Metaphern-Katalog zur Verfügung zu stellen, mit dessen Hilfe er Modelle erstellen kann. In diesem Katalog werden dabei zusätzliche Informationen zu den verwendeten Objektmetaphern hinterlegt, die später auch für die Generierung benutzt werden können.

[30] Benutzerschnittstellenfragmente beschreiben implementierungsnahe Bausteine, aus denen ausführbare Benutzerschnittstellen generiert werden können. Eine Definition und Spezifikation erfolgt in Kapitel 7 dieser Arbeit.



In Abschnitt 5.5 wurde am Beispiel des „Adipositas-Begleiter“ solch ein Katalog eingeführt und dargestellt, wie dieser für die Anwendungsdomäne aufgebaut wird. Die Nutzung solch eines Kataloges für die Modellierung der Interaktionsbeschreibungen schließt schließlich dieses Kapitel ab. Dabei wurde neben dem Modellierungsbeispiel auch ein „Top-Down“-Ansatz für die Modellierung dargestellt, mit dem auch die Modellierung von komplexen Benutzerschnittstellen strukturiert angegangen werden kann.

6 Nutzungskontexte und Nutzungsprofile

In Kapitel 4 wurden im Rahmen der Entwicklung eines Blueprints Modelle und Verfahren definiert, die zur Verfügung zu stellen sind, um eine automatisierte Benutzerschnittstellengenerierung nach dem Leitbild des „Compositional Modeling“ zu ermöglichen. Abbildung 27 greift das in diesem Zusammenhang vorgestellte Modell der Bausteine des Generierungsprozesses (vgl. Abbildung 13) auf und visualisiert (blau hervorgehoben) die Teile, die innerhalb dieses Kapitels erarbeitet werden.

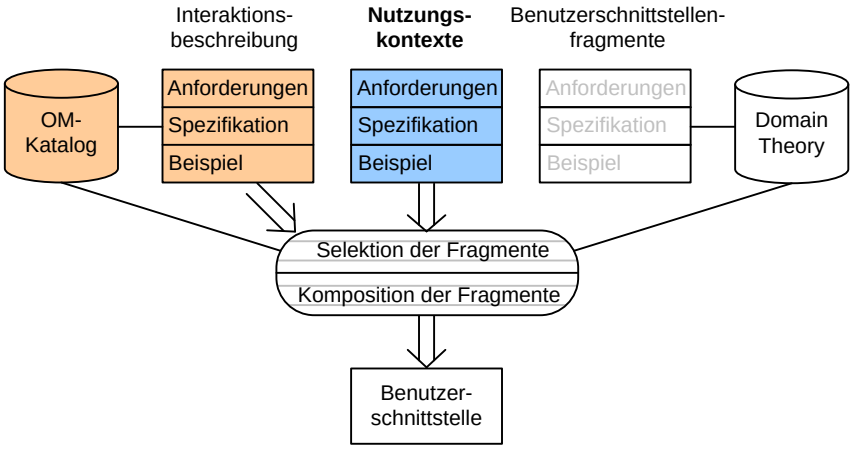


Abbildung 27: Einordnung der Nutzungskontexte in den CM-Prozess

Die Nutzungskontexte werden im Rahmen der Benutzerschnittstellengenerierung als Pendant zu der „query” verwendet, die nach Falkenhainer und Forbus (vgl. [FF91]) folgendermaßen beschrieben wurde:

> *„a query about the scenario's behaviour”*

Im Rahmen des „Compositional Modeling“ definiert die „query“ das eigentliche Ziel des Kompositionsprozesses, nämlich auf welche Fragestellung das generierte Modell Antworten liefern soll.

Im Rahmen der Generierung von „kontextorientierten Benutzerschnittstellen“ wird die Zielsetzung verfolgt, unterschiedliche Rahmenbedingungen zu berücksichtigen. Abhängig von den verfügbaren Endgeräten, den aktuellen Aktivitäten, Präferenzen oder Behinderungen des Nutzers, aber auch von äußeren Einflüssen wie z.B. Lautstärke oder Lichtverhältnissen, sollen effizient unterschiedliche und der Situation angepasste Benutzerschnittstellen erzeugt werden.

Die Modellierung dieser Rahmenbedingungen ist somit eine geeignete Analogie zur „query“ des „Compositional Modeling“. In dieser Arbeit wird ein Kontextmodell herangezogen, um diese Rahmenbedingungen zu erfassen und schließlich soweit zu formalisieren, dass sie für eine automatisierte Generierung herangezogen werden können.

Hierzu wird in einem ersten Schritt der Begriff der „adaptiven Benutzerschnittstelle“ eingehend analysiert, um den sehr weiten Kontextbegriff auf die gegebene Problemstellung zu reduzieren. In einem zweiten Schritt wird ein „generisches Nutzungskontextmodell“ eingeführt, das ein Metamodell für die Beschreibung von „Nutzungskontextmodellen“ darstellt. Ein Nutzungskontextmodell definiert für eine Anwendungsdomäne relevante „Kontextelmente“, mit denen Nutzungskontexte beschrieben werden, die schließlich die Anforderungen an die zu generierende Benutzerschnittstelle kapseln.

Nutzungskontexte bedienen sich einer Beschreibung, welche die Eigenschaften von Benutzern, Endgeräten und der Umgebung der Nutzung erfasst. Um diese Beschreibung für einen automatisierten Generierungsprozess verwenden zu können, wird eine Methodik entwickelt, mit welcher aus einem Nutzungskontext relevante Parameter für die automatisierte Generierung von Benutzerschnittstellen ermittelt werden können. Diese Parameter werden dann in einem so genannten „Nutzungsprofil“ maschinenlesbar gesichert.

Am Beispiel der Instanzierung des „generischen Nutzungskontextmodells“ auf die Anwendungsdomäne des „Adipositas-Begleiters“ wird ein konkretes Nutzungskontextmodell präsentiert, aus dem schließlich Nutzungsprofile abgeleitet werden können.

6.1 Adaptive Benutzerschnittstellen

Adaptive Benutzerschnittstellen werden in der Literatur unterschiedlich definiert. Im Rahmen der Barrierefreiheit ist die Forderung nach adaptiven Benutzerschnittstellen beispielsweise in der „Barrierefreie Informationstechnik-Verordnung“ (vgl. [BITV02]) als Ergänzung des Behindertengleichstellungsgesetzes (vgl. [BGG02]) für öffentlich zugängliche Internetangebote von Behörden der Bundesverwaltung gesetzlich festgeschrieben.

Im Bereich des HCI (Human Computer Interaction[31]) werden unter dem Begriff der Adaptierung von Benutzerschnittstellen zwei unterschiedliche Merkmale verstanden (vgl. [ORK97]): „Adaptive“ Benutzerschnittstellen passen sich selbstständig an die Bedürfnisse des Benutzers an, während „anpassbare“ Benutzerschnittstellen vom Benutzer selbst zumeist über ein entsprechendes Werkzeug modifiziert werden. Aber auch das Verständnis des Begriffes „adaptiv“ ist abhängig von den Zielen, die mit der Adaption der Benutzerschnittstelle verfolgt werden. So definiert Langley in [Lan97] adaptive Benutzerschnittstellen als:

> *„An adaptive user interface is an interactive software system that improves its ability to interact with a user based on partial experiences with that user.”*

[31] HCI steht für eine Disziplin in der Informatik, die sich mit der Evaluation und der Implementierung interaktiver Computersysteme für den Gebrauch durch Menschen und den damit verbundenen Phänomenen beschäftigt (vgl. [ACM92]).

Basierend auf Methoden der künstlichen Intelligenz wird das Verhalten des Nutzers mit dem Ziel analysiert, die Benutzerschnittstelle für den Benutzer zu optimieren. Ein (wenn auch sehr einfaches) Beispiel ist das Menü des Start-Buttons in Windows XP Systemen: hier werden Anwendungen, die der Benutzer oft verwendet, direkt aufgeführt, ohne dass in der Menüstruktur navigiert werden muss, um diese zu finden.

Thevenin und Coutaz (vgl. [TC99]) führen in Zusammenhang mit ihrem Verständnis der Adaptivität von Benutzerschnittstellen ein Klassifikationsschema ein, das Adaptivität durch drei Dimensionen beschreibt:

- Ziel der Adaption
  Diese Dimension beschreibt die Entitäten, für welche eine Adaption durchgeführt werden soll, beispielsweise eine Adaption auf Benutzergruppen, auf Geräteeigenschaften oder an die reale Umgebung der Nutzung (z.B. Lautstärke, Licht, usw.).
- Mittel zur Adaption
  Diese Dimension beschreibt die von der Adaption betroffenen Softwarekomponenten. Beispiele sind die Anpassung der Modalitäten für die Interaktion, die Präsentation von Inhalten, aber auch Veränderungen des Aufgabenmodells. Letzteres kann beispielsweise notwendig sein, wenn die möglichen Aktivitäten des Benutzers mit dem System einschränkt sind, weil kein passendes Endgerät vorhanden ist.
- Temporale Dimension
  Diese Dimension beschreibt schließlich, wann eine Adaption der Benutzerschnittstelle erfolgen soll. Dies kann beispielsweise vor der Nutzung, oder aber auch zur Laufzeit durchgeführt werden.

Basierend auf diesen Dimensionen definieren Thevenin und Coutaz in [TC99] den in der Literatur häufig verwendeten Begriff der „plasticity“ für adaptive Benutzerschnittstellen. Zielsetzung ist die Adaption von Benutzerschnittstellen auf unterschiedliche Endgeräte und die physische Umgebung der Nutzung unter Minimierung der Entwicklungskosten für eine Adaption. Bezogen auf die oben beschriebenen Dimensionen der Adaptivität ist ein „plastic user interface“ nach Thevenin und Coutaz (vgl. [TC99]) durch das folgende Adaptionsziel definiert: Anpassung der Benutzerschnittstelle an die physische Umgebung (Endgerät, Umfeld der Nutzung) unter Verwendung (Mittel zur Adaption) von Eingriffen in die Reihenfolge und Gestaltung der Teilaufgaben und Verwendung von unterschiedlichen Rendering-Techniken. Aussagen über die temporale Dimension werden nicht getroffen.

Plasticity erweitert insbesondere den Begriff der Plattformunabhängigkeit, wie er beispielsweise durch Java (unter Verwendung einer virtuellen Maschine) oder die Verwendung von virtuellen Toolkits (vgl. [Mye95]) definiert wird. Diese ermöglichen zwar die Ausführung der Benutzerschnittstelle auf unterschiedlichen Betriebssystemen oder Softwareplattformen, bieten aber keine Möglichkeit, um Benutzerschnittstellen, die sich physikalisch unterscheiden (kleines Display, andere Modalitäten der Interaktion, Anpassung an äußere Einflüsse wie beispielsweise Umgebungslärm), zu unterstützen.

Der in dieser Arbeit verwendete Begriff der Adaptivität von Benutzerschnittstellen nimmt die Zielsetzung von Thevenin und Coutaz als Ausgangslage, nämlich dass adaptive Benutzerschnittstellen in der Lage sein sollten, sich kosteneffektiv (also möglichst automatisiert) an äußere Rahmenbedingungen anzupassen. Die Dimensionen der Adaptivität, wie sie für die „plastic user interfaces“ beschrieben wurden, erfordern aber entsprechende Erweiterungen:

- Ziel der Adaption
  Eine Adaption soll nicht nur auf physische Rahmenbedingungen (Hardware, äußere Einflüsse) gerichtet sein, sondern wird um zwei weitere Entitäten erweitert: einerseits ist das der Nutzer, seine Möglichkeiten (im Rahmen der Barrierefreiheit) bestimmte Interaktionen durchzuführen, seine aktuellen Aktivitäten (Autofahren, Fernsehen, Arbeit am Desktop-Arbeitsplatz), aber auch seine persönlichen Präferenzen für die Gestaltung von Benutzerschnittstellen. Neben dem Nutzer definiert der Anwendungsanbieter, also der Dienstleister, der dem Benutzer eine Anwendung zur Verfügung stellt, Rahmenbedingungen für die Gestaltung der Benutzerschnittstelle. Eine Benutzerschnittstelle muss also in der Lage sein, sich an unterschiedliche Layoutvorgaben anzupassen. Dies betrifft dabei nicht nur eine Visualisierung im Rahmen eines Corporate Designs (einheitliche Farbgestaltung, Präsentation und Positionierung von Interaktionsobjekten), sondern betrifft auch Navigationsstrukturen, verwendete Interaktionsobjekte, Größe und Organisation von Oberflächenelementen (Fenstergröße, zur Anzeige verfügbarer Bereich usw.).
- Mittel zur Adaption
  Im Gegensatz zu dem modellbasierten Ansatz, den Thevenin und Coutaz verfolgen, liegt kein explizites Aufgabenmodell vor, das als Grundlage für eine Generierung fungieren könnte. Stattdessen existiert ein „abstraktes Interaktionsmodell“ (vgl. Abschnitt „5.1.2 Existierende Interaktionsbeschreibungen“), welches als Werkzeug für die Adaption fungiert. Der Einsatz von Rendering-Techniken für die Darstellung und Transformation ist wie bei den „plastic user interfaces“ ein Mittel zur Adaption.
- Temporale Dimension
  In der vorliegenden Arbeit wird ein statischer Ansatz beschrieben, bei dem der Generierungsprozess vor der Ausführung durchgeführt wird. Änderungen zur Laufzeit sind zwar angedacht aber nicht realisiert, da eine automatische Adaption zur Laufzeit eine vollständige Automatisierung des Generierungsprozesses erfordert, für die zum bisherigen Zeitpunkt zu wenige Erfahrungswerte vorliegen. Eine automatische Generierung ist folglich die nächste Evolutionsstufe des beschriebenen Ansatzes.

Diese Definition der Adaptivität von Benutzerschnittstellen erfordert somit eine Untersuchung und Formalisierung der Kriterien für die Adaption und deren Auswirkungen auf die Benutzerschnittstelle. Diese Arbeiten werden in den folgenden Abschnitten mit dem Ziel durchgeführt, eine formale Beschreibung zu identifizieren, welche Anforderungen für die Generierung liefern.

6.2 Generisches Nutzungskontextmodell

Die Beschreibung der Rahmenbedingungen, die Auswirkungen auf die Generierung von Benutzerschnittstellen haben, ist nicht trivial. Unterschiedlichste Aspekte wie z.B. Endgeräte, Lautstärke der Umgebung, Aktivitäten des Nutzers (z.B. Autofahren) und der Nutzer selbst können Faktoren sein, welche für die Generierung der Benutzerschnittstelle relevant sind. Zwar wurden der Benutzer, die physische Umgebung und der Anwendungsanbieter als solche Entitäten identifiziert (vgl. Abschnitt 6.1), es ist aber nicht ersichtlich, welche Eigenschaften tatsächlich Rahmenbedingungen an die Benutzerschnittstelle definieren, und wie diese unterschiedlichen Aspekte in einem Modell abgebildet werden können.

Ein aktuell vieldiskutierter Ansatz zur Beschreibung von externen Einflüssen auf Softwaresysteme findet sich im „context-aware computing“. Erstmals von Schilit and Theimer in [ST94] im Zusammenhang mit einem mobilen ortsbezogenen Dienst definiert, beschreibt „context-aware computing“ die Erschließung von Informationen wie Ort, Zeit, Situation des Benutzers etc. für den Einsatz in intelligenten und zumeist mobilen Softwaresystemen. Heutzutage kann auf eine Vielzahl von Arbeiten zurückgegriffen werden, die sich mit der Kontextmodellierung, -sensorik und -vorhersage beschäftigen (vgl. [CK00]). Der Ansatz der Kontextmodellierung wird in dieser Arbeit verwendet, um externe Einflüsse auf den Generierungsprozess zu beschreiben.

In den folgenden Abschnitten wird der Begriff des Nutzungskontextes definiert, der den sehr weiten Kontextbegriff nach Dey (vgl. [Dey01]) auf die gegebene Problemstellung reduziert.

6.2.1 Definition von Nutzungskontexten und -profilen

Es sind eine Vielzahl von Definitionen für den Kontextbegriff in der Literatur zu finden. Auch wenn sich keine Definition durchsetzen konnte, liefert die Definition nach Dey [Dey01] eine allgemein akzeptierte Ausgangslage:

> *Context is any information that can be used to characterize the situation of an entity.*
> *An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

Eine Kernaussage dieser Kontextdefinition lautet, dass jede Art der Information ein Teil des Kontextes ist, wenn sie nur Auswirkungen auf die Interaktion von Nutzer und Anwendung hat. Ein häufig genannter Kritikpunkt dieser Definition ist der Mangel an konkreten Ansatzpunkten für die Identifikation von Kontextinformationen. Oftmals findet man Kontextdefinitionen, die Klassifikationen von für den jeweiligen Anwendungsfall relevanten Kontextinformationen enthalten. Verbreitet ist eine Klassifikation, die auf Schilit, Adams und Want in [SAW94] zurückzuführen ist. Folgende Kategorien werden hier benannt:

- Computing context
  beispielsweise Netzwerkverbindung, Kommunikationskosten und –bandbreite, in der Nähe befindliche Ressourcen wie Drucker, Displays oder Rechnersysteme;
- User context
  beispielsweise Nutzerprofile, Personen in der Nähe, augenblickliche soziale Situation;
- Physical context
  beispielsweise Lichtverhältnisse, Lautstärke, Verkehr, Temperatur;

Häufig wird diese Klassifikation um eine vierte Dimension erweitert:

- Time context
  beispielsweise Uhrzeit, Woche, Monat, Jahreszeit.

Die Definition nach Dey und auch die oben aufgeführte Kategorisierung machen deutlich, dass nutzbare Kontextinformationen abhängig vom Verwendungszweck sind.

Daher wird in dieser Arbeit der Begriff des „Nutzungskontextes“ eingeführt. Der Begriff soll veranschaulichen, dass er nur einen Ausschnitt eines allgemeinen Kontextbegriffes darstellt. Erfasst werden nur Informationen, die relevant für die Generierung von Benutzerschnittstellen sind, ohne die Anwendung selbst zu beeinflussen. Damit unterscheiden sich Nutzungskontexte essenziell von dem sonst üblichen Verständnis von kontextsensitiven Anwendungen. Nach Dey (vgl. [Dey01]) zeichnen sich kontextsensitive Anwendungen dadurch aus, dass sie Kontextinformationen nutzen, um dem Nutzer relevante Informationen und Dienste anzubieten, wobei die Relevanz von dem aktuellen Kontext des Nutzers abhängt. Stark vereinfacht kann man also formulieren, dass Kontextinformationen benutzt werden, um Parameter (z.B. Ort, Nutzerprofildaten) automatisch zu ermitteln und der Anwendung bereitzustellen. Da nicht vorhersehbar ist, welche Anwendungen welche Kontextinformationen in welcher Form benötigen, sind oftmals sehr komplexe Kontextmodelle notwendig, um einen gewissen Grad an Generalität zu erhalten. Haseloff zeigt in [Has04], wie komplex Kontextmodelle werden, wenn die Beschreibung auf einer realisierungsnahen Ebene erfolgt.

Nutzungskontexte können einfacher definiert und spezifiziert werden, da sie nur Informationen für und über die Generierung von Benutzerschnittstellen beinhalten und die Anwendungen selbst explizit ausklammern. Ein Nutzungskontext muss in der Lage sein, die folgenden Fragen zu beantworten: Wer nutzt die Schnittstelle? Mit was für einem Endgerät wird die Interaktion durchgeführt und welche Interaktionsmöglichkeiten und Modalitäten können verwendet werden? Welche Einflüsse der Umgebung beeinflussen die Interaktion?

Ein Nutzungskontext setzt sich somit aus drei Komponenten zusammen:

- Informationen über den Nutzer,
- Informationen über das verwendete Endgerät,
- Informationen über die physische Umgebung.

Der Nutzungskontext beinhaltet Informationen, welche direkt (z.B. „Bildschirmgröße des Endgerätes“) oder indirekt (z.B. „Aktivität des Nutzers“) für die Generierung von Bedeutung sind. Da eine automatische Generierung aber immer nur vordefinierte Parameter berücksichtigen kann, wird zusätzlich der Begriff des Nutzungsprofils eingeführt. Ein Nutzungsprofil ist die Konkretisierung eines Nutzungskontextes, der nur explizite Parameter enthält, die für die automatisierte Generierung verwendet werden können.

Im Rahmen des in Abschnitt 4.3 eingeführten Entwicklungsprozesses für die Benutzerschnittstellengenerierung wurden Nutzungskontextmodelle eingeführt, mit denen Nutzungskontexte modelliert werden können. Das Nutzungskontextmodell definiert Modellierungselemente, die auf eine konkrete Anwendungsdomäne ausgerichtet sind, mit denen Nutzungskontexte beschrieben werden können.

In dem folgenden Abschnitt 6.2.2 wird ein generisches Modell (generisches Nutzungskontextmodell genannt) eingeführt, welches die Erzeugung von Nutzungskontextmodellen für unterschiedliche Anwendungsdomänen ermöglicht. Das generische Nutzungskontextmodell wird durch eine Anwendungsdomäne instanziert, für die spezialisierte Modellierungselemente definiert werden, um so ein Nutzungskontextmodell zu spezifizieren.

6.2.2 Generisches Modell eines Nutzungskontextes

Laut Definition setzt sich ein Nutzungskontext aus drei Komponenten (Endgerät, Nutzer, Umgebung) zusammen. Jede der Komponenten beinhaltet Informationen, durch die sie definiert wird. Zur Beschreibung der drei Komponenten wird eine einheitliche Struktur verwendet, die definiert, wie Informationen über die Komponente organisiert sind. Abbildung 28 veranschaulicht das Modell eines Nutzungskontextes, wie es für diese Arbeit definiert wurde, mit Hilfe von Beschreibungselementen eines UML Klassen-Diagramms (vgl. [Bal05]). Objektklassen in dem Diagramm repräsentieren die Entitäten, die für die Beschreibung von Nutzungskontexten verwendet werden.

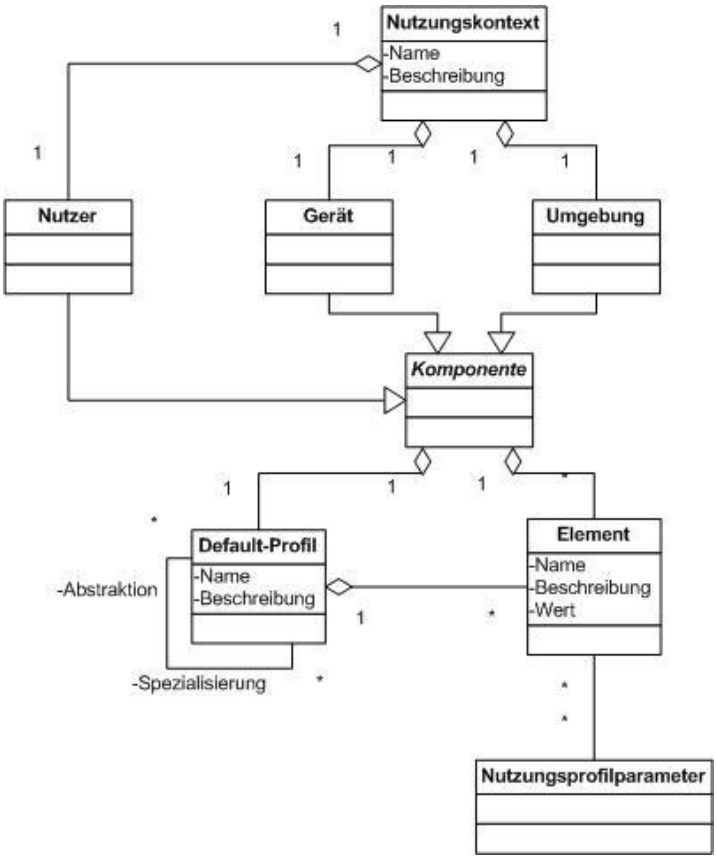


Abbildung 28: Generisches Modell eines Nutzungskontexts

Demnach wird ein Nutzungskontext durch einen Namen, eine Beschreibung und durch die Aggregation der Komponenten Nutzer, Gerät, Umgebung und Anwendungsanbieter beschrieben. Jede der drei Komponenten erbt die Eigenschaften der Klasse Komponente.

Eine Komponente besteht dabei einerseits aus einem Default-Profil und einer Menge von erweiternden Informationen (in Abbildung 28 „Element“ genannt). Das Default-Profil stellt eine Grundmenge von Informationen bereit, die für die Beschreibung der entsprechenden Komponente benötigt werden. Diese Informationsmenge muss Minimalanforderungen genügen, die für eine Generierung der Benutzerschnittstelle gegeben sein müssen. Damit unterscheidet sich dieses Kontextmodell entscheidend von anderen Ansätzen (vgl. [Zim04], [YCJ05]), in denen Kontextinformationen optional sind, und dazu dienen, die Interaktion mit dem Nutzer zu vereinfachen oder zu verbessern. Begründet wird dies zumeist damit, dass wenn Kontextparameter über externe Sensorik erfasst werden, nicht garantiert werden kann, dass valide Informationen immer zur Verfügung stehen. Nutzungskontexte hingegen werden nicht sensorisch erfasst, sondern sind im Vorfeld zu modellieren. Die Verpflichtung, Default-Profile zu definieren, spiegelt die Anforderung wieder, Cluster mit ähnlichen Anforderungen zu bilden und sie mit Informationen zu beschreiben. Jedes Defaultprofil sollte dabei eine gewisse Allgemeingültigkeit für einen Benutzertypus, eine Geräteklasse und ein Nutzungsumfeld aufweisen. Nutzungskontexte sind keine optionalen Informationsquellen, sondern sind essenzieller Bestandteil des Generierungsprozesses.

Default-Profile selbst werden durch einen Namen, eine Beschreibung und eine Menge von „Elementen“ (vgl. „Element“ in Abbildung 28) beschrieben. Ein „Element“ (auch Kontextelement genannt) beschreibt im Bereich der Informationslogistik (vgl. [Has04]) Informationen, über die Kontexte definiert werden können.

Jedem „Element“ werden Nutzungsprofilparameter zugeordnet, die beschreiben, welche Auswirkung das „Element“ auf das Nutzungsprofil hat. Diese Beziehung stellt einen einfachen Mechanismus zur Verfügung, wie auch aus Kontextinformationen für die Generierung nutzbare Parameter identifiziert werden können.

Default-Profile werden dabei hierarchisch strukturiert, wie in Abbildung 28 durch die Relation mit den Enden „Abstraktion“ und „Spezialisierung“ dargestellt wird. Jedes Default-Profil kann somit über eine Abstraktion, also eine Verallgemeinerung seiner selbst, und einer Menge von Spezialisierungen verfügen. Dieses Konstrukt beschreibt den Ansatz, allgemeine Default-Profile durch bestimmte Kriterien zu differenzieren, die eine Aufspaltung in Sub-Defaultprofile ermöglichen. Sub-Defaultprofile können durch andere oder Verfeinerung von bereits angewendeten Kriterien weiter unterteilt werden.

Die Komponenten von Nutzungskontexten (Endgerät, Nutzer, Umgebung) werden neben den Default-Profilen durch erweiternde Informationsbausteine beschrieben. So kann jedes Default-Profil durch zusätzliche Informationen verändert oder erweitert werden. In Abbildung 28 beschreiben die mit „Element“ gekennzeichneten Rechtecke solche Informationsbausteine. Dieses Konstrukt wurde eingeführt, um die Komplexität der Hierarchien, die für die Default-Profile erstellt werden, beherrschbar zu machen, und andererseits auch die Möglichkeit zu geben, flexibel Anpassungen durchzuführen, die sich klassischerweise nicht in Hierarchien abbilden lassen.

6.3 Nutzungsprofile

Im Gegensatz zum Nutzungskontextmodell, das an die jeweilige Domäne angepasst bzw. neu instanziert werden muss, sind Parameter eines Nutzungsprofils (vgl. Abschnitt 4.3) durch die Möglichkeiten zur Adaption der Benutzerschnittstelle eindeutig definiert. Einschränkungen können aus unterschiedlichen Gründen vorgenommen werden: verantwortlich kann der Mensch sein, der nicht in der Lage ist, bestimmte Interaktionen durchzuführen, oder die Umgebung, in der sich der Nutzer befindet, aber natürlich auch das Endgerät mit seinen Interaktionsmöglichkeiten.

Während Nutzungskontextmodelle ein Werkzeug zur Hand geben, um für eine Anwendungsdomäne angepasste äußere Einflüsse zu modellieren und zu bewerten, beschreibt das Nutzungsprofil die konkreten Auswirkungen auf die Benutzerschnittstelle. Nutzungsprofile beschreiben ausschließlich Präferenzen für Modalitäten, Medien und Interaktionsformen der Benutzerschnittstelle.

6.3.1 Beschreibung von Nutzungsprofilen

Im Rahmen der Adaptivität von Anwendungen an Geräteeigenschaften im Umfeld des Pervasive Computing[32] sind eine Reihe von Profildefinitionen wie beispielsweise das Composite Capabilities/Preference Profiles (kurz CC/PP) (vgl. [Kiss07]), User Agent Profile (kurz UAProf) (vgl. [OMA06]) oder das Wireless Universal Resource File (kurz WURFL) (vgl. [WURF]) entstanden.

Das vom W3C entwickelte CC/PP hat sich dabei weitgehend etabliert. Insbesondere im Bereich der Mobilfunktelefone und Smart Phones sind für viele am Markt verfügbare Endgeräte entsprechende Profile zu finden. CC/PP sollte originär ein Geräteprofil beschreiben, das im Web bei jedem Request an einen Server gesendet wird, der dann basierend auf dem Profil entsprechende Inhalte aufbereiten konnte. Das User Agent Profile (kurz UAProf) (vgl. [OMA06]) ist die am weitesten verbreitete Variante des CC/PP. UAProf ist ein auf CC/PP basierender Standard, der von der Open Mobile Alliance (ehemals WAP Forum) gefördert wird, und ursprünglich für den Einsatz mit dem Wireless Application Protocol (kurz WAP) (vgl. [WAP]) geplant war.

Das CC/PP Profil basiert auf dem Resource Description Framework (kurz RDF) (vgl. [BB04]). CC/PP Profile sind in Form eines zweistufigen Baumes definiert. Die Wurzel bildet die Kennung des CC/PP Profils, das über beliebig viele Komponenten verfügen kann. Jede der Komponenten kann schließlich durch eine Menge von Attributen beschrieben werden. Diese Struktur spiegelt die RDF-Semantik wieder, indem Aussagen über Komponenten (Subjekt) getroffen werden, die über benannte Attribute (Prädikate) beschrieben werden, denen schließlich Werte (Objekte) zugeordnet sind. Somit ist CC/PP als Vokabular für die Beschreibung von Profilen zu verstehen, mit dem spezialisierte Profile erstellt oder existierende erweitert werden können. Ein vorgegebenes Parameterset definiert UAProf, das mit Hilfe des CC/PP Vokabulars erweitert werden kann. Hier werden folgende Komponenten für die Beschreibung von Geräten identifiziert:

- Hardware platform
- Software platform
- WAP characteristics
- Browser user agent
- Network characteristics

[32] Pervasive Computing ist ein häufig verwendeter Begriff für Anwendungen, die über unterschiedliche zumeist mobile Endgeräte verfügbar gemacht werden. Während Ubiquitous Computing immer noch als Vision der Mensch-Maschine Interaktion ohne die Wahrnehmung von Computern verstanden wird, versteht sich das Pervasive Computing als Zwischenschritt, der sich durch vernetzte mobile Endgeräte auszeichnet und damit den heutigen Stand der Technik repräsentiert.

CC/PP wird dabei nicht nur für die Beschreibung von Geräteprofilen verwendet. Es finden sich einige Ansätze in der Literatur (vgl. [IRR*03]), die ausgehend von einem UAProf Erweiterungen vorschlagen, die Kontextinformationen wie Ort oder Umgebungsinformationen beinhalten. Als Resümee dieser Arbeiten kann zusammenfasst werden, dass die größte Problemstellung die einfache Struktur von CC/PP Profilen darstellt. Kontextmodelle benötigen die Möglichkeit, komplexere Zusammenhänge und Beziehungen zu modellieren, die sich nur mit „Kunstgriffen“ in CC/PP überführen lassen. Eine Übertragung dieser Kontextmodelle auf CC/PP ist somit durchführbar, aber nicht empfehlenswert, da die Beschreibungen zunehmend unüberschaubarer werden, so dass die resultierenden CC/PP Profile nicht mehr von Menschen zu lesen sind.

Diese Argumentation widerspricht nicht dem im Folgenden beschriebenen Ansatz, CC/PP als Mittel zur Beschreibung von Nutzungsprofilen zu verwenden. Nutzungsprofile beschreiben laut Definition für die Generierung relevante Parameter. Im Gegensatz zu komplexen Kontextmodellen ist eine einfache Struktur ausreichend, in der für Benutzerschnittstellen relevante Entitäten strukturiert und parametrisiert werden. Die Entkopplung von Kontextinformationen von maschinenverarbeitbaren Profilinformationen ermöglicht hier den Einsatz von CC/PP für die Beschreibung der letzteren, ohne dadurch auf die semantischen Informationen, die in dem Kontextmodell hinterlegt sind, zu verzichten.

Ausgangslage für die Beschreibung der Spezifikation des Nutzungsprofils bildet das UAProf. Hier werden bereits Parameter für die Beschreibung der Hardware der verwendeten Endgeräte ausreichend detailliert abgebildet. Da ein Nutzungskontext weit über die reine Beschreibung von Endgeräten, deren Softwarecharakteristiken und Konnektivitätseigenschaften hinausgeht, werden in den folgenden Abschnitten notwendige Erweiterungen auf Basis von CC/PP vorgestellt. Die eingeführten Erweiterungen umfassen Aspekte der Benutzerschnittstelle, sowie eine detaillierte Beschreibung physischer Interaktionsmedien bis zu Interaktionsobjekten graphischer Benutzerschnittstellen, Layoutvorgaben und der Beschreibung der Strukturierung einer Benutzerschnittstelle. Strukturiert werden diese (wie in UAProf üblich) über so genannte Komponenten.

UAProf nutzt XML zur Beschreibung der Komponenten und Parameter. Aus Gründen einer Verbesserung der Lesbarkeit wird auf eine XML-Beschreibung in den folgenden Kapiteln verzichtet. Stattdessen wird (wie oftmals in CC/PP üblich) eine Baumstruktur zur Beschreibung der Parameter verwendet.

6.3.2 Input-Widget Komponente

Die Input-Widget Komponente wurde neu eingeführt und erweitert UAProf durch Beschreibungen von Interaktionsobjekten für Benutzerschnittstellen. UAProf selbst bietet lediglich eine indirekte Beschreibung von möglichen Interaktionsobjekten durch die Spezifikation der Software- und Browser-Komponenten, die auf dem Gerät vorinstalliert sind. Diese legen aber ausschließlich generelle Rahmenbedingungen fest und definieren, ob ein Gerät überhaupt in der Lage ist, bestimmte Interaktionsobjekte darzustellen, indem beschrieben wird, welche Software und welche Browser auf dem Gerät verfügbar

sind. Ob ein Benutzer in der Lage ist, diese Interaktionsobjekte zu bedienen und welche Präferenzen hierbei zu beachten sind, wird aber nicht bestimmt.

Die Input-Widget Komponente abstrahiert von den soft- und hardwaretechnischen Möglichkeiten des Gerätes und definiert für unterschiedliche Interaktionsobjekte nicht nur, ob, sondern auch inwieweit diese Interaktionsobjekte zu benutzen sind.

Input-Widget					
→	GUI-Widgets				
	→	Button			
		→	Preference	0-9	
		→	Image-Label	0-9	
		→	Text-Label	0-9	
			→	Text-Size-Small	0-9
			→	Text-Size-Medium	0-9
			→	Text-Size-Large	0-9
	→	Tree			
		→	Preference	0-9	
		→	Max-Tree-Level	1-5	
		→	Browsable	0-9	
		→	Text-Label	0-9	
		→	Image-Label	0-9	
	→	Text-Link	...		
	→	Menue	...		
	→	Check-List	...		
	→	Radio-Button	...		
	→	Throttle	...		
	→	Link-List	...		
	→	Drop-Down-List	...		
	→	Text-Field	...		
	→	Text-Box	...		
→	Vocal-Widges				
	→	Voice-Comand	...		
	→	Speech-to-Text	...		
	→	Voice-Recording	...		

Tabelle 3: Ausschnitt der Input-Widget Komponente

Tabelle 3 stellt die Struktur der Input-Widget Komponente dar. Unterschieden wird dabei, ob die Interaktionsobjekte eine graphische Repräsentation besitzen (diese werden unter GUI-Widgets (vgl. Tabelle 3) gelistet) oder ob sie alternative Modalitäten, wie beispielsweise Sprache oder Gesten verwenden. In Tabelle 3 sind im Verzeichnis „Vocal-Widgets“ Interaktionsobjekte der Modalität Sprache gelistet. „Voice-Command“ beschreibt dabei die Verwendung von Sprachkommandos zur Steuerung der Benutzerschnittstelle, während „Speech-to-Text“ die Möglichkeit beschreibt, Spracheingaben als Text zu interpretieren und „Voice-Recording“ schließlich die Möglichkeit zur Aufzeichnung von Sprachnachrichten.

Jedes Interaktionsobjekt wird durch eine Reihe von Parametern beschrieben. Preference ist dabei ein Standardparameter, der immer Teil der Beschreibung ist und angibt, mit welcher Präferenz dieses Interaktionsobjekt einzusetzen ist. Dabei wird ein Zahlenwert von 0 bis 9 (0: nicht einsetzbar, 9: zu bevorzugen) eingesetzt. Weitere Parameter sind optional und dienen dazu, Konfigurationen des Interaktionsobjektes zu bewerten. Auch hier wird ein Wert von 0 bis 9 eingesetzt, um zu beschreiben, mit welcher Präferenz diese Konfigurationsmöglichkeit zu benutzen ist.

Tabelle 3 stellt einen repräsentativen Ausschnitt an Interaktionsobjekten und ihren Parametern dar. Die Parameter vom Interaktionsobjekt Button und Tree veranschaulichen exemplarisch mögliche Konfigurationen der beiden Interaktionsobjekte. So können Buttons beschreibende Bilder oder Texte beinhalten, wobei die verwendeten Textgrößen ebenfalls ein möglicher Parameter sein können. Navigationsbäume (in Tabelle 3 Tree genannt) besitzen Parameter wie beispielsweise „maximale Tiefe“ des Baumes (in Tabelle 3 Max-Tree-Level genannt), ob sich die Zweige des Baumes öffnen und wieder schließen lassen (in Tabelle 3 Browsable genannt), und schließlich, wie die Knoten des Baumes dargestellt werden (in Tabelle 3 Text-Label, bzw. Image-Label). Die verwendeten Interaktionsobjekte und deren Parameter orientieren sich dabei an existierenden Beschreibungssprachen für Benutzerschnittstellen. Insbesondere wurde UIML (vgl. [APA04]), USIXML (vgl. [LVM*05]) und Teresa XML (vgl. [MPS04]) für die Ermittlung relevanter Interaktionsobjekte herangezogen.

6.3.3 Output-Widget Komponente

Die Output-Widget Komponente bildet das Gegenstück zu der Input-Widget Komponente, die aber Benutzerschnittstellenelemente beschreibt, die ausschließlich der Ausgabe von Informationen dienen, ohne dass sie irgendwelche Interaktionsmechanismen anbieten. Die Struktur und Semantik ist analog zur Input-Widget Komponente definiert. Tabelle 4 veranschaulicht einen repräsentativen Ausschnitt möglicher Ausgabeelemente und deren Parameter. Analog zu der Output-Widget Komponente bildeten existierende Beschreibungssprachen für Benutzerschnittstellen die Grundlage für die identifizierten Elemente.

Output-Widget				
→	GUI-Widgets			
	→	Text		
		→	Preference	0-9
		→	Text-Size-Small	0-9
		→	Text-Size-Medium	0-9
		→	Text-Size-Large	0-9
		→	Colorset-B/W	0-9
		→	Colorset-Easy	0-9
		→	Colorset-Normal	0-9
	→	Picture	..	
	→	Grid	...	
→	Audio-Widget			
	→	Speach	...	
	→	Text-to-Speach	...	

	→	Notification-Sounds	...	
	→	Background-Music	...	
→	Multimedia			
	→	Video	...	
	→	...	...	

Tabelle 4: Ausschnitt der Output-Widget Komponente

6.3.4 Erweiterung der Hardware-Plattform Komponente

Die Hardware-Plattform Komponente, wie sie in UAProf definiert ist, beschreibt eine Reihe von Parametern für die Beschreibung der Hardware-Eigenschaften von Endgeräten. Darunter fallen beispielsweise Bildschirmgröße, Eingabemedien (Tastatur, Sondertasten usw.), Eingabemöglichkeiten (Ziffern, Buchstaben usw.) und auch die Darstellungsmöglichkeiten von Bild und Multimediadaten. Erweiterungen dieser Komponente sind notwendig, um Präferenzen abbilden zu können. Schließlich wurde eine genauere Betrachtung der physischen Interaktionsmöglichkeiten integriert. So wird in UAProf gewöhnlich nur verzeichnet, dass eine Tastatur verfügbar ist, nicht jedoch aus welchen Einzelblöcken (z.B. Letters, Numbers, Number-pad, Arrows) solch eine Tastatur besteht und wie diese einzelnen Möglichkeiten zu bewerten sind. In der Tabelle 5 wird ein repräsentativer Ausschnitt der Hardware-Plattform Komponente abgebildet.

„Hardware-Plattform“ Komponente				
	Input			
		Keyboard		
			Preference	0-9
			Letters	0-9
			Numbers	0-9
			Number-pad	0-9
			Arrows	0-9
		Mouse		
			Preference	0-9
			OnScreen-Keyboard	0-9
			Drag&Drop	0-9
			Left-Click	0-9
			Right-Click	0-9
			Scroller	0-9
			Special	0-9
		Touch-Pad	...	
		Touch-Display		
			Preference	0-9
			OnScreen-Keyboard	0-9
			...	
		Phone-Keypad	...	
		PDA-ControlKey	...	

		Remote-Control	...	
		Voice-Control	...	

Tabelle 5: Erweiterung der Hardware Komponente

6.3.5 Layoutpattern Komponente

Während die anderen drei eingeführten Komponenten (Input-, Output-Widget und Hardware-Plattform) im wesentlichen Benutzerschnittstellenelemente oder physische Interaktionsobjekte mit entsprechenden Präferenzen für deren Nutzung beschreiben, liefert die Layoutpattern Komponente mögliche Strukturierungen der Oberfläche der Benutzerschnittstelle.

Der Begriff des Layoutpatterns basiert auf der von Luyten, Coninx und Abrams in [LCA05] eingeführten Definition. Ein Layoutpattern ist somit analog zu einem Raumplan aufgebaut, der beschreibt, wo sich bestimmte Räume befinden und wie diese Räume aufgebaut sind. Ein Layoutpattern für Benutzerschnittstellen beschreibt nun Bereiche der Benutzerschnittstelle, in der Darstellungen realisiert sind, oder Bereiche, denen bestimmte Aufgaben zugeordnet wurden. Abbildung 29 visualisiert exemplarisch ein Layoutpattern.

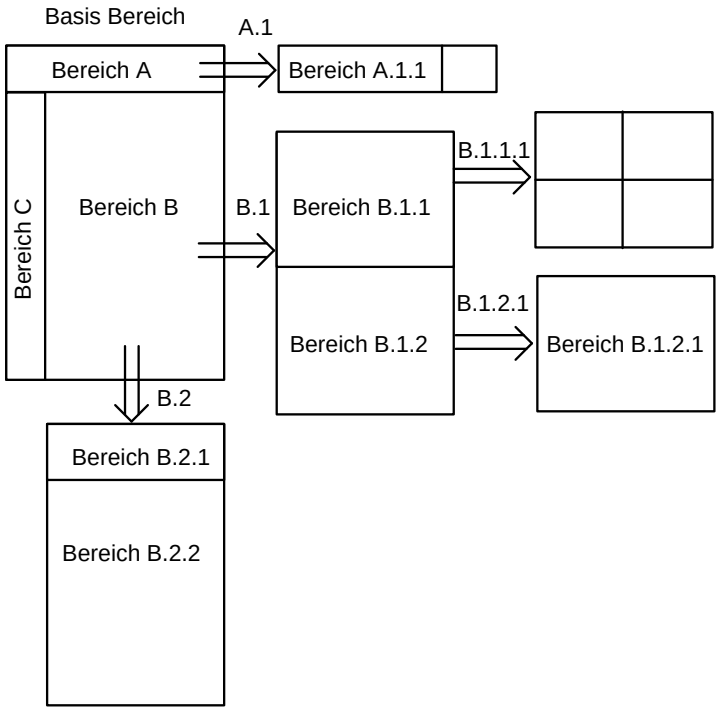


Abbildung 29: Exemplarisches Layoutpattern

Das große Rechteck (links oben in Abbildung 29) repräsentiert eine graphische Oberfläche, die in drei Bereiche (Bereich A, B und C) eingeteilt ist. Jeder dieser Bereiche definiert eine bestimmte Raummenge und eine Position innerhalb der Benutzerschnittstelle und kann unterschiedliche Interaktionsobjekte beherbergen. Die in Abbildung 29 dargestellten Kanten stellen nun Erweiterungen des Layoutpattern-

Ansatzes von Luyten, Coninx und Abrams (vgl. [LCA05]) dar, die nun mögliche Varianten für die Strukturierung der im Basis-Bereich (Rechteck links oben in Abbildung 29) definierten Bereiche beschreiben. So kann beispielsweise der Bereich B in zwei Varianten (B1 und B2 in Abbildung 29) weiter strukturiert werden. Im Gegensatz zur Definition von Luyten, Coninx und Abrams (vgl. [LCA05]), die auf einen Pool von möglichen Layoutpattern verweisen, aus denen genau eines auszuwählen ist, wird durch die eingeführten Varianten eine Menge von Möglichkeiten definiert, die bei der Generierung der Benutzerschnittstelle verwendet werden können. Der Vorteil dieses Vorgehens wird insbesondere bei komplexen Anwendungen offensichtlich, die unterschiedliche Sub-Dienste nutzen, wie sie beispielsweise bei Internet-Portalen zu finden sind. Hier findet man oftmals eine Basisstruktur, die sich entsprechend der Interaktion des Nutzers in unterschiedliche Varianten unterteilt. Abhängig von der Komplexität der Anwendung und deren Platzbedarf auf der Oberfläche können hier unterschiedliche Varianten herangezogen werden.

Die Layoutpattern Komponente definiert nicht nur ein Pattern, sondern vielmehr eine Menge von Pattern-Varianten, die für die Generierung eingesetzt werden können. Dabei sind die einzelnen Varianten nicht mit Präferenzen versehen. Welche Variante für die Generierung herangezogen wird, hängt im Wesentlichen von den darzustellenden Interaktionsobjekten ab, die aber erst im Laufe des Generationsprozesses bestimmt werden können. Ein Layoutpattern wird durch die einzelnen Bereiche und deren unterschiedliche Ausgestaltung definiert. Jeder Bereich wird durch seine absolute Position auf der Oberfläche definiert und durch weitere Parameter, wie beispielsweise, ob der Bereich scrollbar ist, beschrieben. Tabelle 6 veranschaulicht ein Beispiel eines Layoutpatterns.

Layoutpattern				
	Base-Area			
		Top-Left	0/0	
		Down-Right	160/220	
		Scrollbar	N	
		Dynamic	N	
		Mandatory	Y	
		Sub	<Sub-A>	
		Sub	<Sub-B>	
		Sub	<Sub-C>	
	Sub-A			
		Top-Left	0/0	
		Down-Right	20/220	
		Scrollbar	N	
		Dynamic	Y	
		Mandatory	Y	
		Sub	<Sub-A.1>	
	Sub-B			
		Top-Left	21/0	
		Scrollbar	Y	
		Down-Right	160/20	
		Mandatory	Y	

		Sub	<Sub-B.1>	
		Sub	<Sub-B.2>	
...	...	...	...	
	Dynamic-Windows	Preference	0-9	
	Voice			
		Hierarchical	Preference	0-9
		Command	Preference	0-9

Tabelle 6: Layoutpattern

Neben der Beschreibung der Bereiche beinhaltet das Layoutpattern zusätzliche Informationen über die Benutzerschnittstelle, beispielsweise ob und welche der Bereiche als dynamische Oberflächenbereiche geöffnet werden können. Benutzerschnittstellen bieten oftmals die Möglichkeit, weitere Oberflächenbereiche zu öffnen (beispielsweise durch das Öffnen neuer Fenster).

Das Prinzip des Layoutpatterns kann auch auf andere Benutzerschnittstellentypen übertragen werden. So könnten beispielsweise sprachgesteuerte Benutzerschnittstellen über eine gewisse Strukturierung beschrieben werden. In Tabelle 6 werden Layoutpattern für zwei sprachbasierte Benutzerschnittstellen definiert. Im ersten Pattern wird über eine hierarchische Menüstruktur navigiert, während beim zweiten Pattern mit speziellen Sprachkomandos gearbeitet wird. Letztere Variante wird beispielsweise für die Steuerung von Desktop-Oberflächen verwendet, die sich nicht auf eine einfache hierarchische Struktur reduzieren lassen.

Da Bereichen auch eine Zweckbindung zugesprochen wird, indem jeder Bereich einer bestimmten Aufgabe zugeordnet ist, beeinflusst das Layoutpattern die Benutzerschnittstellengestaltung in einem hohen Maße. So definiert ein Layoutpattern mit wenigen großen Bereichen einen vorwiegend sequentiellen Interaktionsablauf, während komplexe Layoutpattern (siehe Abbildung 29) eine Vielzahl von Informationen parallel darstellen können und somit höchst komplexe und dynamische Interaktionsfolgen ermöglichen.

Neben der logischen Verbindung von Bereichen existiert auch eine physische Abhängigkeit. Bereiche verfügen über einen gewissen Umfang und legen einen definierten Rahmen für die maximale Größe von graphischen Interaktionsobjekten, Schriftgrößen und Abständen zwischen den Interaktionsobjekten fest. Eine entsprechende Größe ist aber notwendig, um Benutzerschnittstellen beispielsweise für Sehbehinderte oder auch Menschen mit motorischen Störungen anbieten zu können. Tabelle 7 veranschaulicht die Layoutpattern Komponente.

Layoutpattern Komponente				
	Interaction flow			
		sequential	Preference	0-9
		parallel	Preference	0-9
		efficient	Preference	0-9
		mutable	Preference	0-9
	Layout			
		simple	Preference	0-9
		complex	Preference	0-9
	Dynamic windows	Preference	0-9	
	Scrollbars	Preference	0-9	

Tabelle 7: Layoutpattern Komponente

Die Layoutpattern Komponente liefert Informationen zur Auswahl eines Layoutpatterns, wobei die Elemente mit folgender Semantik hinterlegt sind:

- „Interaction flow“
  „Sequential“ beschreibt die Präferenz für einen sequentiellen Interaktionsablauf, wie er beispielsweise über Wizards realisiert wird. Indikator für ein Layoutpattern, das solch einen Ablauf unterstützt, sind wenige und möglichst große Bereiche.
  „Parallel“ beschreibt die Präferenz für eine hohe Informationsdichte auf der Benutzerschnittstelle mit vielen Bereichen, die gleichzeitig vom Benutzer bearbeitet werden können.
  „Efficient“ beschreibt die Präferenz für eine schnelle Interaktionsdurchführung, die in möglichst wenigen Interaktionsschritten abgeschlossen ist.
  „Mutable“ beschreibt die Präferenz für alternativenreiche Benutzerschnittstellen, in denen unterschiedliche Wege zur Zielerreichung angeboten werden.
- „Layout“
  „Simple“ beschreibt die Präferenz für eine einfache Layout-Strukturierung der Benutzerschnittstelle. Als Indikator für solch eine Strukturierung gilt ein geringes Maß an Variationen (geringe Anzahl von Subbereichen) im Layoutpattern. Die Tiefe der Subbereiche ist ebenfalls ein Indikator für die Komplexität des Layoutpatterns.
- „Dynamic windows“
  Präferenz der Nutzung von sich dynamisch öffnenden Bereichen (z.B. Fenstern).
- „Scrollbar“
  Präferenz der Nutzung von Scrollbars.

6.4 Nutzungskontexte und -profile für den Adipositas-Begleiter

In den vorhergehenden Abschnitten wurden generische Modelle und auch Spezifikationen erarbeitet, mit deren Hilfe relevante Parameter für die Adaptivität von Benutzerschnittstellen modelliert werden können. Zur Modellierung dieser Parameter wurde ein einfaches Kontextmodell gewählt, und darauf aufbauend ein Nutzungskontextmodell definiert, das es ermöglicht, die für Benutzerschnittstellen relevanten Kontextkomponenten (Benutzer, Endgerät und Umgebung) abzubilden. Das Nutzungskontextmodell selbst wird durch die Kontextkomponenten und diese wiederum durch Kontextelemente beschrieben. Welche Kontextelemente für die Beschreibung Verwendung finden, ist abhängig von der Anwendungsdomäne.

In den folgenden Abschnitten wird am Anwendungsbeispiel des „Adipositas-Begleiters“ (vgl. Abschnitt 3.3) ein konkretes Kontextmodell erarbeitet, indem Default-Profile für jede der drei Kontextkomponenten und schließlich die notwendigen Kontextelemente benannt werden. Dabei wird nur ein kleiner Ausschnitt möglicher Kontextelemente beschrieben, die aber bereits einen für den Anwendungsbereich „Adipositas-Begleiter“ ausreichenden Umfang aufweisen. Der gewählte Ausschnitt erhebt weder einen Anspruch auf Vollständigkeit, noch ist er repräsentativ für die Anwendungsdomäne. Beschrieben werden lediglich Ergebnisse von Workshops, die in diesem Zusammenhang mit den Therapeuten der Gelderland-Klinik erarbeitet wurden.

In einem zweiten Schritt wird anhand des entwickelten Kontextmodells vorgestellt, wie aus den Kontextelementen, die einen Nutzungskontext beschreiben, ein Nutzungsprofil abgeleitet wird. Im Rahmen dieser Betrachtung werden drei Nutzungskontexte und die daraus generierten Nutzungsprofile präsentiert, die im späteren Teil der vorliegenden Arbeit zur Evaluation des Generierungsprozesses herangezogen werden.

6.4.1 Kontextkomponente Endgerät

Kontextelemente der Komponente „Endgerät“ beschreiben physische Eigenschaften von Endgeräten wie beispielsweise Displaygröße, Verbindungseigenschaften, Inputmodalitäten, aber auch softwaretechnische Aspekte, wie Betriebssysteme und verfügbare Anwendungen.

Auch wenn die Benennung von Kontextelementen in diesem Bereich möglich ist, kann auf eine Modellierung dieser Elemente verzichtet werden, da die Eigenschaften durch das verwendete Endgerät (beschrieben durch Hersteller, Gerätetyp und Gerätenummer) vollständig spezifiziert sind. Die Kontextkomponente Endgerät nimmt damit eine Sonderstellung im Rahmen der Kontextmodellierung ein, da die Hersteller bereits ausreichend Informationen in Form von Endgeräteprofilen (beschrieben in UAProf) anbieten. Da UAProf aber auch Bestandteil des Nutzungsprofils ist, können diese Informationen direkt in das Nutzungsprofil übernommen werden.

Im Rahmen des Anwendungsszenarios Adipositas-Begleiter wurden zusammen mit den Therapeuten und Patienten Geräteklassen identifiziert, die für den Einsatz als Adipositas-Begleiter interessant sein können. Abbildung 30 visualisiert relevante Gerätekategorien aus diesem Bereich.

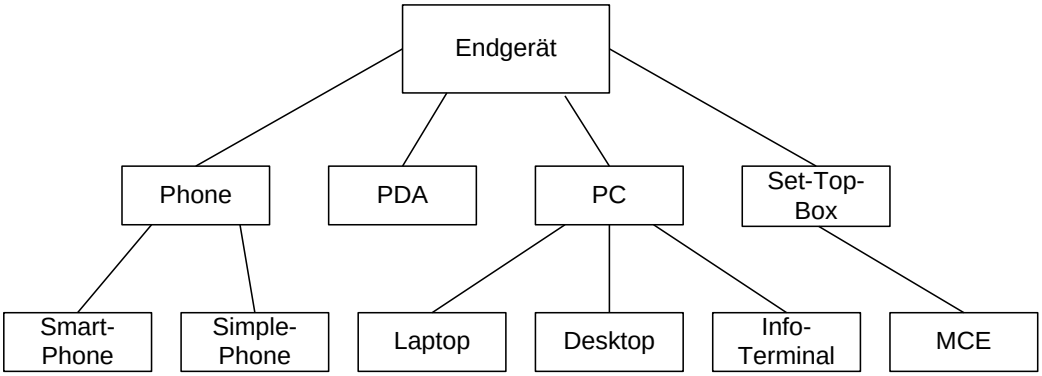


Abbildung 30: Hierarchische Dekomposition der Kontextkomponente „Endgerät“

Diese Kategorisierung kann benutzt werden, um Gerätebeschreibungen zu definieren, die in einem gewissen Rahmen generisch sind. So definiert jede Geräteklasse eine Menge von Eigenschaften, die alle realen Endgeräte, die dieser Klasse zugeordnet sind, realisieren können.

Zusätzlich zu den Endgerätekategorien können für die Modellierung der Komponente „Endgerät“ erweiternde Kontextelemente wie beispielsweise Tastatur, Maus, Kopfhörer und Mikrofon eingesetzt werden.

So kann beispielsweise das MCE[33] ausschließlich über eine Fernbedienung gesteuert werden. Zusätzlich hat der Benutzer die Möglichkeit eine Tastatur und/oder eine Maus anzuschließen. Dies erweitert natürlich die Interaktionsmöglichkeiten bzw. vereinfacht Texteingaben erheblich.

6.4.2 Benutzer

Während die Hard- und Softwareplattform größtenteils generisch beschrieben werden kann, ist das Benutzermodell stark von der Anwendungsdomäne abhängig. Somit erfordert die Definition des Benutzermodells eine genauere Betrachtung der Zielgruppe, deren Bedürfnisse und Auswirkungen auf die Benutzerschnittstellen.

Grundlage für das hier verwendete Modell der Benutzerkomponente sind Betrachtungen, die im Umfeld des „Adipositas-Begleiters“ (vgl. Abschnitt 3.3) zusammen mit den Fachärzten, Therapeuten und Patienten gemacht wurden. Zielsetzung dieser Betrachtung war eine Clusterbildung, die Patienten mit ähnlichen Vorlieben und Bedürfnissen zusammenfasst.

[33] MCE (Microsoft Media Center Edition) repräsentiert an dieser Stelle den Einsatz von Set-Top-Boxen, die mit Hilfe einer Fernbedienung zu bedienen sind.

Aus Befragungen und Expertenworkshops konnten drei Patientenprofile ermittelt werden, die sich im Wesentlichen durch den Umgang und die Erwartungen an den „Adipositas-Begleiter“ unterscheiden. Es ist dabei zu bemerken, dass die Profile höchst spezifisch auf das im Abschnitt 3.3 dargestellte Anwendungsszenario (Patienten der Gelderland-Klinik nach einer stationären Therapie) ausgerichtet sind, und daher weder den Anspruch auf Vollständigkeit noch Allgemeingültigkeit haben können. Die Profile sind jedoch ausreichend, um für das Anwendungsbeispiel „Adipositas-Begleiter“ die Generierung von Benutzerschnittstellen zu ermöglichen.

Beschrieben werden die Profile anhand eines Bezeichners und einer kurzen informellen Beschreibung der Eigenschaften der Personen, die in diesem Profil zusammengefasst werden. Spezifiziert wird das Profil durch eine Menge aus Kontextelementen. Um die Menge unterschiedlicher Kontextelemente zu beschränken existiert jedes Kontextelement in den vier Varianten ++, + , -, --. Wobei gilt:

- ++ definiert eine **sehr positive** Auswirkung des Kontextelements auf das Nutzungsprofil;
- + definiert eine **positive** Auswirkung des Kontextelements auf das Nutzungsprofil;
- - definiert eine **negative** Auswirkung des Kontextelements auf das Nutzungsprofil;
- -- definiert eine **sehr negative** Auswirkung des Kontextelements auf das Nutzungsprofil;

Es sei darauf hingewiesen, dass keine „neutrale“ Präferenz existiert. Neutral bedeutet, dass weder positive noch negative Auswirkungen auf das Nutzungsprofil zu erwarten sind. Eine Modellierung dieses Kontextelementes ist daher auch nicht notwendig.

- *Name*: **Technikaffine Nutzer**
  Beschreibung: Menschen, die Gefallen an dem Umgang mit interaktiver Software haben und Interesse entwickeln, die Möglichkeiten zu erproben. Sie zeichnen sich durch einen spielerischen Umgang mit dem Softwaresystem aus und sind bereit, durch „Ausprobieren“ die Anwendung zu erkunden.
  Kontextelemente:
  - Präferenzen für symbolische und graphische Darstellungen; (++)
  - Präferenzen für die Nutzung von multimedialen Inhalten; (+)
  - Präferenzen für eine flexible Aufgabendurchführung; (++)
  - Präferenzen für komplexe, interaktive Interaktionsobjekte und Interaktionen; (++)
  - Präferenzen für hohe Informationsdichte auf der Benutzerschnittstelle; (++)
  - Präferenz für die Nutzung von dynamischen Fenstern; (++)
- *Name*: **Nutzer**
  Beschreibung: Menschen, die eine feste Vorstellung haben, welche Ziele die Anwendung verfolgt und über Erwartungen verfügen, wie diese zu erreichen sind. Im Vordergrund steht ein klarer und effizienter (schneller, insbesondere ohne Wiederholungen) Interaktionsablauf.
  Kontextelemente:

- Präferenz für „Maus und Tastatur“; (++)
- Präferenzen für symbolische und graphische Darstellungen; (+)
- Präferenz für eine „effiziente Aufgabendurchführung“; (++)
- Präferenzen für komplexe, interaktive Interaktionsobjekte und Interaktionen; (-)
- Präferenz für hohe Informationsdichte auf der Benutzerschnittstelle; (++)
- Präferenz für die Nutzung von dynamischen Fenstern; (+)

- *Name*: **Nicht technikaffine Nutzer**
  Beschreibung: Menschen, die eine Führung durch den Interaktionsablauf erwarten. Sie erwarten klare, sich wiederholende Strukturen und benötigen ein Feedback des Systems über getätigte Interaktionen.
  Kontextelemente:
  - Präferenzen für beschriftete Interaktionsobjekte; (++)
  - Präferenz für die Nutzung von multimedialen Inhalten; (+)
  - Präferenzen für eine sequentielle Aufgabendurchführung; (+)
  - Präferenzen für die Texteingaben über die Tastatur (++);
  - Präferenzen für einfache, nicht interaktive Interaktionsobjekte und Interaktionen; (+)
  - Präferenzen für komplexe, interaktive Interaktionsobjekte und Interaktionen; (--)
  - Präferenzen für hohe Informationsdichte auf der Benutzerschnittstelle; (-)
  - Präferenzen für Sondertasten (speziell Steuertasten und Tastenkombinationen); (-)
  - Präferenz für die Nutzung von dynamischen Fenstern; (--)

Neben den über die Patientenprofile beschriebenen Kontextelementen sind für das Anwendungsbeispiel „Adipositas-Begleiter“ zwei weitere Kontextelemente zu berücksichtigen, die variabel jedem der drei Patientenprofile zugeordnet werden können.

- Sehvermögen; (++, +, -, --)
  Beschreibt die Fähigkeit zur Wahrnehmung der Benutzerschnittstelle, wobei „++“ ein sehr gutes Sehvermögen modelliert. Das Sehvermögen hat die folgenden Auswirkungen auf das Nutzungsprofil:
  - Schriftgröße und Größe der Beschriftung von Interaktionsobjekten;
  - Informationsdichte auf der Benutzerschnittstelle;
  - Präferenz von Interaktionsobjekten, die eine vergrößerte Darstellung von Inhalten ermöglichen, wie sequentieller Interaktionsfluss, Scrollbars und dynamische Fenster;
  - Wahl des Farblayouts;
  - Bei einem sehr eingeschränkten Sehvermögen sollten Audio-Ausgaben präferiert werden;

- Motorik; (++, +, -, --)
  Beschreibt die Fähigkeit zur Durchführung von Interaktionen, wobei „++“ sehr gute motorische Fähigkeiten modelliert. Die Motorik hat die folgenden Auswirkungen auf das Nutzungsprofil:
  - Einsatz von „Pointing Devices“ (bspw. Maus, Touch-Display usw.) sollte bei stark eingeschränkten motorischen Fähigkeiten vermieden werden;
  - Einsatz von Tastaturen (auch Fernbedienungen) ist bei eingeschränkten motorischen Fähigkeiten zu bevorzugen;
  - Menge von Interaktionsobjekten auf der Benutzerschnittstelle sollte gering sein;
  - Nutzung komplexer Interaktionen (Drag&Drop, Pull-down-Menus, Scrollbars);
  - Größe (und damit auch Erreichbarkeit) der Interaktionsobjekte;
  - Bei eingeschränkten motorischen Fähigkeiten sollte eine Sprach-Steuerung präferiert werden;

6.4.3 Umfeld der Nutzung

Das Umfeld der Nutzung beschreibt physische aber auch soziale Faktoren, die für die Interaktion von Bedeutung sind. Typische Merkmale der physischen Umgebung, wie Lärmpegel, Lichtverhältnisse, aber auch aktuelle Aufmerksamkeit des Benutzers oder Personen im Umfeld sind Teil dieser Kontextkomponente. Für den Adipositas Begleiter wurde eine einfache Kategorisierung in drei für den Anwendungsfall relevante Bereiche gegliedert. Dabei sind die gleichen Einschränkungen für die Übertragbarkeit und Allgemeingültigkeit des vorgestellten Modells zu berücksichtigen, wie sie bereits im Abschnitt 6.4.2 „Benutzer“ dargestellt wurden.

- *Name*: **Unterwegs**
  Beschreibung: „Unterwegs“ beschreibt ein Umfeld, in dem der Benutzer mobil ist, wie beispielsweise als Fußgänger, Bahnfahrer aber auch Autofahrer. Der Benutzer ist weitgehend abgelenkt und zumeist von Menschen umgeben.
  Kontextelemente:
  - Störung durch Umgebungslärm; (+)
  - Präferenz für eine sequentielle Aufgabendurchführung; (++)
  - Präferenzen für beschriftete Interaktionsobjekte; (++)
  - Präferenzen für komplexe, interaktive Interaktionsobjekte und Interaktionen; (-)
  - Präferenz für die Nutzung von dynamischen Fenstern; (--)
- *Name*: **Arbeit**
  Beschreibung: „Arbeit“ beschreibt ein Umfeld, in dem der Adipositas-Begleiter und damit auch die Aufmerksamkeit der Benutzerschnittstelle nur eine untergeordnete Rolle spielt. Interaktionen sollten schnell und effektiv abgearbeitet werden können, so dass ein konzentriertes Arbeiten mit dem Adipositas-Begleiter möglich ist, auch wenn es sich nur über eine sehr kurze Periode erstrecken sollte.

Einzelne Arbeitschritte sollten jederzeit unterbrechbar ein und nach einer Unterbrechung einen schnellen Einstieg in die Anwendung ermöglichen.
Kontextelemente:

- Störung durch Umgebungslärm; (+)
- Präferenz für eine „effiziente Aufgabendurchführung“; (+)
- Präferenz für die Nutzung von multimedialen Inhalten; (--)

- *Name*: **Zuhause**
  *Beschreibung: „*Zuhause“ beschreibt Umgebungen, in denen der Benutzer einen Großteil seiner Konzentration dem Adipositas Begleiter widmen und sich auch über eine längere Periode mit ihm auseinander setzen kann.
  Kontextelemente:
  - Störung durch Umgebungslärm; (-)
  - Präferenz für die Nutzung von multimedialen Inhalten; (+)

Neben den über die Umgebungsprofile beschriebenen Kontextelementen sind für das Anwendungsbeispiel „Adipositas-Begleiter“ drei weitere Kontextelemente zu berücksichtigen, die variabel jedem der drei Patientenprofile zugeordnet werden können.

- Lichtverhältnisse (++, +, -, --)
  Beeinflussen die Lesbarkeit von Schriften und die Wahrnehmung der Benutzerschnittstelle beispielsweise bei starker Sonnenstrahlung, wobei „++“ sehr gute Lichtverhältnisse modelliert. Die Lichtverhältnisse haben die folgenden Auswirkungen auf das Nutzungsprofil:
  - Präferenz für große Schriften und Interaktionsobjekte;
  - Präferenz für ein einfaches Farb-Layout;
  - Präferenz für eine geringe Informationsdichte der Benutzerschnittstelle;
  - Präferenzen für Audio-Ausgaben;
  - Vermeidung von multimedialen Inhalten;
- Personen im Umfeld
  Personen im Umfeld sind ein Indikator, dass ein Benutzer sich nur teilweise auf die Interaktionen konzentrieren kann und gegebenenfalls den Interaktionsfluss unterbrechen muss, wobei „++“ Personen im Umfeld modelliert, welche die Benutzerschnittstelle einsehen können. Personen im Umfeld haben die folgenden Auswirkungen auf das Nutzungsprofil:
  - Vermeidung komplexer Interaktionen (Drag&Drop, Pull-down-Menus, Scrollbars);
  - Präferenz für eine sequentielle Aufgabendurchführung;

- Ablenkungen

  Dieses Kontextelement legt fest, wie viel Aufmerksamkeit ein Nutzer einer Benutzerschnittstelle schenken kann. Eingeschlossen ist damit auch die Konzentrationsdauer, die definiert, wie lange komplexe Interaktionsschritte dauern sollten, bis ein Status erreicht ist, der eine Unterberechung ermöglicht. Eine starke Ablenkung des Benutzers wird durch „++“ modelliert. Die Ablenkungen des Benutzers haben die folgenden Auswirkungen auf das Nutzungsprofil:

  - Vermeidung komplexer Interaktionen (Drag&Drop, Pull-down-Menus, Scrollbars);
  - Präferenz für eine sequentielle Aufgabendurchführung;

6.4.4 Generierung von Nutzungsprofilen für den Adipositas-Begleiter

In Abschnitt 6.3 wurde die Struktur von Nutzungsprofilen definiert, in welcher für die Generierung der Benutzerschnittstelle notwendige Informationen abgelegt werden. Parametrisiert wird das Nutzungsprofil nur indirekt durch die Definition von Nutzungskontexten. Es ist nicht vorgesehen, dass das Nutzungsprofil manuell erstellt wird. Dies ist zwar möglich, es erfordert aber ein tiefes technisches Verständnis über Endgeräte, deren Eigenschaften, aber auch über die Auswirkungen von Eigenschaften von Benutzern und deren Umgebung auf die Benutzerschnittstelle.

Nutzungskontextmodelle wurden daher als Werkzeug eingeführt, um auf einer abstrakteren Ebene Nutzungsprofile definieren zu können. Ein Nutzungskontextmodell ist auf eine Anwendungsdomäne hin angepasst und beschreibt für die Anwendungsdomäne typische Einflussfaktoren (gegeben durch „Default-Profile“ und Kontextelemente) auf die Benutzerschnittstelle. Es stellt somit einen Baukasten bereit mit dem Nutzungskontexte beschrieben werden können.

Ein Nutzungskontext definiert konkrete Rahmenbedingungen (in Form von numerischen Parametern), die für die Anpassung der Benutzerschnittstelle relevant sind. Insbesondere wird darin beschrieben, welche Art von Nutzern, welches Endgerät mit welcher Soft- und Hardwareumgebung und auch welche äußeren Einflüsse bei der Generierung der Benutzerschnittstelle zu berücksichtigen sind. Die Strukturierung der einzelnen Kontextkomponenten des Nutzungskontextmodells ist dabei analog zueinander definiert (vgl. Abschnitt 6.2). Eine Komponente wird durch Kontextelemente beschrieben. In den Kontextelementen werden wiederum die konkreten Eigenschaften erfasst. Eine Klassifikation von für die Anwendungsdomäne relevanten Stellvertretern (beispielsweise Personengruppen mit ähnlichen Eigenschaften) wird eingesetzt, um für die Anwendungsdomäne typische Nutzungskontexte generieren zu können. Ein Nutzungskontext wird modelliert, indem für jede Komponente ein entsprechender Stellvertreter (Default-Profile) ausgewählt und gegebenenfalls durch zusätzliche Kontextelemente erweitert wird.

Abbildung 31 veranschaulicht die Beschreibung eines Nutzungskontextes und das Vorgehen für die Ermittlung des zugehörigen Nutzungsprofils. Das hellgraue Rechteck auf der rechten Seite in Abbildung 31 visualisiert einen Nutzungskontext, beschrieben durch die Komponenten Nutzer, Gerät und Umgebung.

Jede der Komponenten wird durch einen Stellvertreter (in Abbildung 31 mit <Default> gekennzeichnet) und durch verfeinernde Kontextelemente (in Abbildung 31 mit $Element_{1,}$ ..., $Element_3$ gekennzeichnet) beschrieben. Sowohl dem Stellvertreter als auch jedem Kontextelement sind Parameter für Nutzungsprofile, wie sie in Abschnitt 6.3 definiert wurden, zugeordnet. Der jeweilige Stellvertreter der Komponente beinhaltet dabei eine relevante Grundmenge an Parametern für Nutzungsprofile, die sich aus den beschreibenden Kontextelementen ermitteln lässt.

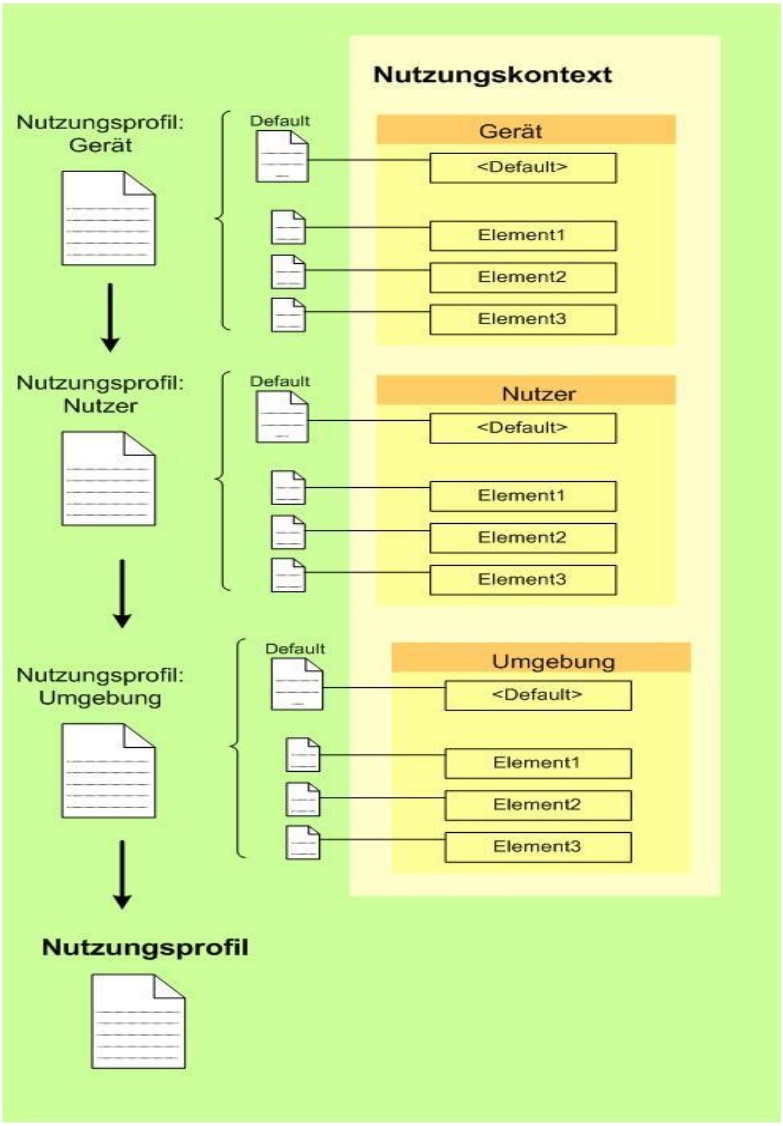


Abbildung 31: Ermittlung eines Nutzungsprofils aus einem Nutzungskontext

Der Generierungsprozess des Nutzungsprofils beginnt mit der Analyse der Kontextkomponente „Gerät“. Diese spezifiziert insbesondere das Endgerät und definiert damit einen Rahmen, welche Interaktionen und Benutzerschnittstellenelemente überhaupt angeboten werden können. Damit beschreibt diese Komponente eine Vorbelegung des gesamten Nutzungsprofils, welches in den folgenden Schritten durch die anderen Komponenten (Nutzer, Umgebung) modifiziert wird.

Das Wissen über Kontextelemente und deren Auswirkung auf das Nutzungsprofil ist im Nutzungskontextmodell hinterlegt. Jedem Kontextelement wird dabei eine Menge von Elementen des Nutzungsprofils zugeordnet. Die folgende Notation zur Beschreibung eines Kontextelementes wird verwendet:

<Name des Kontextelementes>
<Beschreibung>

<Kompontente$_1$>.<Profilname$_1$>:(<Default$_1$>):<Modifikator$_1$>

<Kompontente$_2$>.<Profilname$_2$>:(<Default$_2$>):<Modifikator$_2$>

...

<Kompontente$_n$>.<Profilname$_n$>:(<Default$_n$>):<Modifikator$_n$>

Das Kontextelement „Sehvermögen (-)“ aus dem Anwendungsbeispiel des „Adipositas-Begleiters“ ist wie folgt definiert:

Sehvermögen (-)
Beschreibt die Fähigkeit zur Wahrnehmung der Benutzerschnittstelle. Wobei „++“ ein sehr gutes Sehvermögen modelliert.

Input-Widgets.Vocal-Widgets.Voice-Command.Preference: (6) +1
Input-Widgets.Vocal-Widgets.Speech-to-Text.Preference: (6) +1
Input-Widgets.Vocal-Widgets.Voice-Recording.Preference: (6) +1
Input-Widgets.GUI-Widgets.Button.Text-Label.Text-Size-Small: (3) -2
Input-Widgets.GUI-Widgets.Button.Text-Label.Text-Size-Medium: (5) +0
Input-Widgets.GUI-Widgets.Button.Text-Label.Text-Size-Large: (7) +2
Input-Widgets.GUI-Widgets.Tree.Text-Label.Text-Size-Small: (3) -2
Input-Widgets.GUI-Widgets.Tree.Text-Label.Text-Size-Medium: (5) +0
Input-Widgets.GUI-Widgets.Tree.Text-Label.Text-Size-Large: (7) +2
...
Output-Widgets.GUI-Widges.Text.Text-Size-Small: (3) -2
Output-Widgets.GUI-Widges.Text.Text-Size-Medium: (5) 0
Output-Widgets.GUI-Widges.Text.Text-Size-Big: (7) +2
Output-Widgets.GUI-Widges.Text.Colorset-Easy: (7) +2
...
Layoutpattern.Interaktions-flow.sequential.Preference: (7) +2
Layoutpattern.Interaktions-flow.sequential.Preference: (7) +2
Layoutpattern.Interaktions-flow.sequential.Preference: (7) +2

Layoutpattern.Layout.simple. Preference: (7) +2

Layoutpattern.Layout.complex. Preference: (5) +0

Dieser Ausschnitt für die Definition des Kontextelementes veranschaulicht, dass für jeden Parameter sowohl ein Default-Wert als auch ein Modifikator angegeben wird. Der Default-Wert wird bei der Generierung des Nutzungsprofils herangezogen, falls der Parameter im Nutzungsprofil bisher nicht vertreten war. Ansonsten wird der im Nutzungsprofil vorhandene Wert durch den Modifikator verändert.

Im Folgenden werden exemplarisch drei Nutzungskontexte beschrieben, die im späteren Verlauf der Arbeit für die Evaluation des Generierungsprozesses herangezogen werden.

		Youngster unterwegs mit dem PDA	Best Ager zuhause am Fernseher	Mid Ager bei der Arbeit am PC
Endgerät	Default	Apple IPhone 3G	Set-Top-Box - Tastatur	PC
Nutzer	Default	technikaffiner Nutzer - Sehvermögen: ++ - Motorik: +	nicht technikaffiner Nutzer - Sehvermögen: o - Motorik: -	Nutzer - Sehvermögen: + - Motorik: +
Umgebung	Default	Unterwegs - Lichtverhältnisse: + - Personen im Umfeld: - - Ablenkung: +	Zuhause - Lichtverhältnisse: ++ - Personen im Umfeld: + - Ablenkung: -	Arbeit - Lichtverhältnisse: ++ - Personen im Umfeld: - - Ablenkung: +

Tabelle 8: Nutzungskontexte für den Anwendungsfall „Adipositas-Begleiter“

Tabelle 8 veranschaulicht den Aufbau von Nutzungskontexten. Jede Spalte beschreibt genau einen Nutzungskontext, wobei die Spaltenüberschrift den Namen des Kontextes angibt, und jede Zeile eine der Kontextkomponenten beschreibt. Definiert wird eine Kontextkomponente durch die Angabe eines entsprechenden Default-Profils (beschriftete Rechtecke in Tabelle 8), wie sie in den Abschnitten 6.4.1, 6.4.2

und 6.4.3 für den „Adipositas-Begleiter“ eingeführt wurden. Neben dem Default-Profil kann durch die Angabe von zusätzlichen Kontextelementen die Kontextkomponente erweitert werden.

Es ist leicht darstellbar, dass bereits mit den wenigen Kontextelementen, die für den „Adipositas-Begleiter“ definiert wurden, eine große Menge an möglichen Nutzungskontexten und damit auch Nutzungsprofilen beschrieben werden können. Für die Evaluation des Generierungsprozesses wurde versucht, möglichst klar voneinander abgrenzbare Nutzungskontexte zu wählen, die zu stark variierenden Benutzerschnittstellen führen sollen.

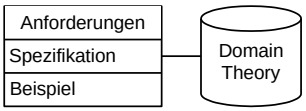
Die den drei Nutzungskontexten zugeordneten Nutzungsprofile finden sich in Anhang B.

6.5 Zusammenfassung und abschließende Betrachtung

Im Rahmen des Generierungsprozesses stellt das Nutzungsprofil die zentrale Informationsquelle für die Beschreibung der Anforderungen dar, denen die generierte Benutzerschnittstelle genügen muss. In Abschnitt 6.3 wurde die Struktur von Nutzungsprofilen definiert. Aufgebaut sind diese Profile analog zum UAProf (als CC/PP Profile), das auf Basis von RDF (Resource Description Framework) um weitere Komponenten erweitert wurde. Betrachtet man den Umfang eines Nutzungsprofils, so liegt die Problemstellung nahe, dass eine manuelle Erstellung solcher Profile mit erheblichem personellem Aufwand verbunden wäre, und weiterhin aufgrund der Menge der Parameter auch schwer zu interpretieren ist.

Um dieser Problemstellung zu begegnen wurde im Rahmen der Anforderungsmodellierung von Benutzerschnittstellen ein Kontextmodell zur Modellierung von Nutzungskontexten eingeführt (vgl. Kapitel 6.2.2). Nutzungskontexte beschreiben auf der Basis von Kontextelementen die drei für die Generierung von Benutzerschnittstellen relevanten Komponenten Endgerät, Nutzer und Umgebung der Nutzung. Mit deren Hilfe ist es möglich auf einer abstrakten Ebene solche Anforderungen zu modellieren, um in einem weiteren Schritt automatisiert daraus Nutzungsprofile abzuleiten. Im Rahmen des im Abschnitt 4.3 beschriebenen Entwicklungsprozesses ist die Unterstützung bei der Erstellung von Nutzungsprofilen ein wichtiger Bestandteil der Entwicklung, da für eine möglichst große Anzahl an Nutzungskontexten Benutzerschnittstellen generiert werden sollen.

Anhand des Anwendungsbeispiels „Adipositas-Begleiter“ konnte weiterhin gezeigt werden, dass bereits mit Hilfe einfacher Nutzungskontextmodelle, die nur einige wenige Kontextelemente umfassen, eine Vielzahl verschiedener Nutzungskontexte generiert werden kann. Darüber hinaus wurden drei Nutzungskontexte entwickelt und vorgestellt, welche die Basis für die Evaluation des Generierungsprozesses sein werden.



7 Benutzerschnittstellenfragmente und Domain-Theory

In Kapitel 4 wurden im Rahmen der Entwicklung eines Blueprints Modelle und Verfahren definiert, die zur Verfügung zu stellen sind, um eine automatisierte Benutzerschnittstellengenerierung nach dem Leitbild des „Compositional Modeling“ zu ermöglichen. Abbildung 32 greift das in diesem Zusammenhang vorgestellte Modell der „Bausteine des Generierungsprozesses“ (vgl. Abbildung 13) auf und visualisiert (blau hervorgehoben) die Teile, die innerhalb dieses Kapitels erarbeitet werden.

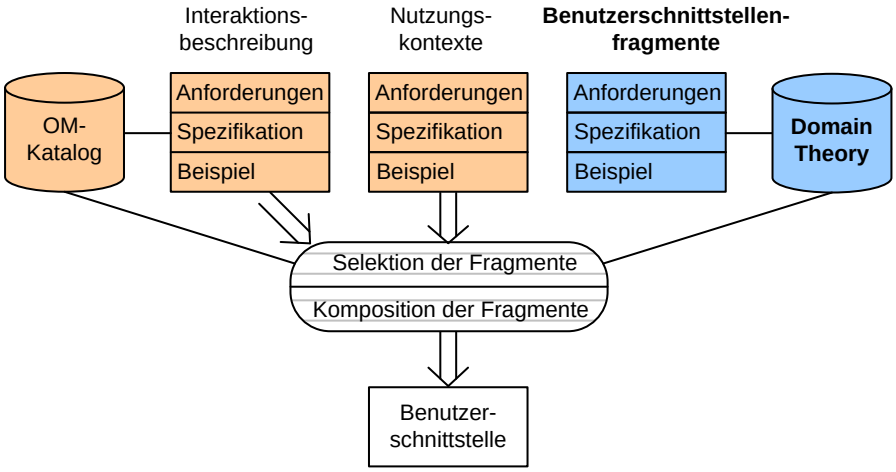


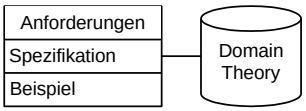
Abbildung 32: Einordnung der Benutzerschnittstellenfragmente in den CM-Prozess

Innerhalb dieses Kapitels werden die „Domain Theory“ und damit einhergehend insbesondere die Benutzerschnittstellenfragmente (kurz BSF) als Pendant zu den „domain model fragments“ (beim „Compositional Modeling“) erarbeitet. Die „Domain Theory“ wird von Falkenhainer und Forbus (vgl. [FF91]) folgendermaßen beschrieben:

> *“a „Domain Theory“ Th, consisting of a set of domain model fragments {m_1 , . . . , m_n} and a set of rules constraining their use”*

Weiterhin sagen sie in [FF91], dass eine „Domain Theory“ eine Klasse von in Beziehung stehenden Phänomenen oder Systemen beschreibt. „Domain model fragments“ sind nach Falkenhainer und Forbus Bestandteile wie Prozesse, Bauteile und Objekte und repräsentieren eine Wissensbasis, die durch Komposition zu neuem Wissen führen soll.

Die Übertragung des Begriffes „Domain Theory“ in das Umfeld der Benutzerschnittstellengenerierung muss dabei die Fragestellung beantworten, wie eine Wissensbasis in diesem Bereich modelliert werden kann. Der Ansatz, der dabei in dieser Arbeit verfolgt wird, sieht vor, tatsächliche Realisierungen von



Benutzerschnittstellen als Wissensbasis zu nutzen. Realisierung heißt dabei nicht, dass es sich um kompilierten Source-Code handeln muss, sondern dass eine Spezifikation existiert, die direkt in ausführbare Benutzerschnittstellen überführt werden kann.

In Analogie zu dem in der Definition von Falkenhainer und Forbus verwendeten Begriff der „domain model fragments“ werden in diesem Kapitel Benutzerschnittstellenfragmente spezifiziert, die wieder verwendbare Teile von Benutzerschnittstellen und ihre Möglichkeit zur Komposition beschreiben. Diese kapseln das Wissen um die Realisierung der Benutzerschnittstelle, beinhalten aber auch Wissen, wie sich diese verhalten und welche Schnittstellen sie anbieten.

Hierzu werden in einem ersten Schritt Anforderungen an Benutzerschnittstellenfragmente formuliert, die sie erfüllen müssen, um diese komponieren und schließlich daraus auch ausführbare Benutzerschnittstellen generieren zu können. Basierend auf dieser Anforderungsanalyse erfolgen die Konzeption und die Beschreibung einer Spezifikationsmethodik für Benutzerschnittstellenfragmente.

Darauf aufbauend wird eine „Domain Theory“ für die Benutzerschnittstellengenerierung definiert und am Beispiel des Adipositas-Begleiters präsentiert.

7.1 Anforderungen an Benutzerschnittstellenfragmente

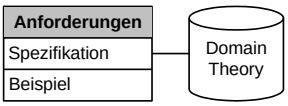
Benutzerschnittstellen beschränken sich nicht nur auf die graphische Präsentation. Sie bilden auch die dynamischen Aspekte ab, wie die Veränderung der Benutzerschnittstelle nach Interaktionen des Nutzers und/oder der Daten, die von einer Applikation bereitgestellt werden. Das Seeheimmodell (vgl. [CCN97]), das im Rahmen der UIMS in dieser Arbeit im Abschnitt „5.1.4.1 Interaktionsmodellierung mittels UIMS“ eingeführt wurde, benennt Komponenten und deren Aufgaben:

a) Die *Präsentation* realisiert die graphische Repräsentation von Daten und Interaktionsobjekten;

b) Die *Dialogsteuerung* nimmt Benutzereingaben entgegen und leitet diese an die Anwendungsschnittstelle weiter. Zusätzlich nimmt sie Daten aus der Anwendungsschnittstelle entgegen und modifiziert die Präsentation;

c) Die *Anwendungsschnittstelle* nimmt Daten aus Benutzerinteraktionen entgegen und liefert Inhalte für die Präsentation;

Das Seeheimmodell als Architekturmodell für interaktive Systeme beschreibt die wesentlichen funktionalen Teile von Benutzerschnittstellen. So lassen sich die drei funktionalen Bausteine auch in anderen Benutzerschnittstellenmodellen wieder finden, wie beispielsweise in MVC (Model View Controller) (vgl. [Mye98]) oder auch das auf Agenten basierende PAC (Presentation, Abstraction, Control) (vgl. [CCN97]). Betrachtet man beispielsweise das MVC Pattern genauer, so werden hier ebenfalls drei Teilbereiche von Anwendungen unterschieden:



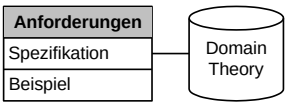
- Das Model, das den Kern der Anwendungsdaten und Basisfunktionalitäten beschreibt.
- Die View, die Daten aus dem Model bezieht und eine Präsentation für den Benutzer realisiert.
- Der Controller, der Daten aus Benutzerinteraktionen bezieht, diese an das Model weiterreicht und mit Daten aus dem Model die View manipuliert.

Lässt man nun Softwarearchitekturaspekte außen vor, wie beispielsweise, dass im Seeheimmodell die Dialogsteuerung (analog der Controller im MVC) ihre Daten über die Präsentation erhält, während im MVC der Controller diese Daten direkt bezieht, so ist die Einteilung der Funktionalitäten in beiden Modellen analog zueinander definiert. Um eine einheitliche Benennung der drei Aspekte von Benutzerschnittstellen zu gewährleisten, wird im Folgenden ausschließlich dem Seeheimmodell entsprechend von Präsentation, Dialog und Anwendungsschnittstelle gesprochen. Dabei sollte nicht vernachlässigt werden, dass die Realisierung dieser drei Bereiche in unterschiedlichen Arten von Benutzerschnittstellen (Web-Schnittstellen, Tabellenkalkulation oder CAD-Tools) auf verschiedene Weise erfolgen kann.

Typische Desktop-Benutzerschnittstellen, wie sie mit den Microsoft Foundation Classes, Java-Swing oder tcl/tk realisiert werden, nutzen einen Event-basierten Ansatz für die Dialogbeschreibung. Die Präsentation wird über Toolkits realisiert, mit Hilfe dessen vordefinierte Benutzerschnittstellenobjekte (Fenster, Buttons, Listen) erzeugt werden können. Interaktionen mit diesen Elementen lösen Events aus, die zu einer Modifikation der Präsentation, aber auch zum Zugriff auf die Anwendungsschnittstelle führen können. Die Anwendungsschnittstelle wird durch den Aufruf dieser Module realisiert, die eine interne Verarbeitung kapseln.

Web-Anwendungen auf der anderen Seite nutzen HTML für die Realisierung der Präsentation, während Java-Servlets, Perl oder Microsoft Active Pages für die Realisierung der Dialogsteuerung verwendet werden, die selbst den Zugriff auf Datenbanken oder externe Funktionen realisiert. Dabei werden diese Werkzeuge (Java-Servlets, Perl usw.) sowohl für die Beschreibung des Dialoges, als auch für die Beschreibung der internen Anwendungslogik verwendet.

Anhand dieser beiden Beispiele (Desktop-Benutzerschnittstellen und Web-Anwendungen) wird deutlich, dass auch wenn die Benutzerschnittstelle durch die drei Bereiche beschrieben werden kann, eine Trennung zum übrigen Teil der Anwendung zwar durch die Anwendungsschnittstelle definiert, aber zumeist nicht realisiert wird. Insbesondere die Trennung von Dialog und Anwendungsschnittstelle gestaltet sich als problematisch, da hier Dialogabläufe mit anwendungsinternen Abläufen synchronisiert werden müssen. In der Softwareentwicklung wird versucht, diese Trennung durch Architekturvorgaben durchzusetzen, die zumeist in Form von Frameworks angeboten werden. Beispiele für solche Frameworks sind JavaStruts (vgl. [Weß06]), Sun J2EE (JAVA) und das Microsoft .NET Framework. Eine automatische Generierung von Benutzerschnittstellen ist nur möglich, wenn beide Bereiche klar voneinander getrennt betrachtet und auch bearbeitet werden können.



Die folgenden Abschnitte betrachten die Anforderungen an die drei Bereiche Präsentation, Dialog und Anwendungsschnittstelle, welche die Grundlage für die Beschreibung und schließlich die Spezifikation von Benutzerschnittstellenfragmenten liefern.

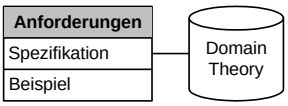
7.1.1 Anforderungen an die Präsentation

Auf den ersten Blick ist die Aufgabe der Präsentation vollständig durch die Realisierung der Darstellung von Benutzerschnittstellen beschrieben. Übersehen wird dabei, dass sie nicht nur die Darstellung von Daten realisiert, sondern auch Elemente bereitstellt, über welche der Benutzer mit dem System interagiert. Somit definiert die Präsentation auch die Interaktionsmöglichkeiten des Nutzers. Weiterhin realisiert die Präsentation das direkte „Interaktions-Feedback“ an den Benutzer. Wenn ein Button angeklickt wird, verändert er sich kurzzeitig auf der Präsentation und gibt dadurch dem Benutzer für die Betätigung des Buttons ein graphisches Feedback. Dabei kann die Präsentation auch in der Lage sein, kleine Interaktionsabläufe zu integrieren. Beispiele für solche Interaktionsabläufe sind Scrollbars, Pull-down-Menüs, Editieren in einem Textfenster, Auswählen von Checkboxes usw. In all diesen Beispielen führt ein Benutzer mehrere Aktivitäten mit diesen Elementen durch, ohne dass erwartet wird, dass hier bereits eine Dialogsteuerung eingreifen müsste.

Eine klare Trennung, welche Interaktionsmöglichkeiten eine Präsentation beinhaltet und ab wann eine Dialogsteuerung benötigt wird, ist von der Realisierung der Präsentation und der Laufzeitumgebung abhängig. So können auch hoch komplexe Oberflächenobjekte (auch Widgets genannt) existieren, die vielfältige und auch komplexe Interaktionsmöglichkeiten bieten (bspw. ein Map-Widget, das die Darstellung und den Umgang mit einer Weltkarte realisiert). Im Gegensatz dazu bieten HTML-Seiten laut Spezifikation nur eine ganz beschränkte Anzahl an Widgets, die nicht erweitert werden können. Die Realisierung von komplexen Interaktionsmöglichkeiten auf Ebene der Präsentation scheint von besonderem Interesse zu sein, wie die Erweiterung von HTML durch Java-Skript aufzeigt. Java-Skript bietet die Möglichkeit, Interaktionen auf der Präsentation zu realisieren, ohne dass ein Server-System in Aktion treten muss.

Die Präsentation stellt weiterhin den plattformabhängigen Teil der Benutzerschnittstelle dar. Der Begriff der Plattform beschreibt in dieser Arbeit einerseits die Abhängigkeit einer Benutzerschnittstelle vom verwendeten Endgerät (Desktop-PC, PDA, Smart Phone), andererseits aber auch von unterschiedlichen Laufzeitumgebungen von Benutzerschnittstellen, wie beispielsweise Browser, Windows-Anwendung, Java-Anwendung oder auf Eclipse basierende Benutzerschnittstellen.

Eine Wiederverwendung auf Ebene der realisierten Benutzerschnittstelle (Programm-Code, compilierter Code) kann keinen gangbaren Weg darstellen. Auch der Einsatz von Softwarekomponenten bietet nur Bausteine, die zwar für die Implementierung, aber nicht für eine automatische Generierung als wieder verwendbare Bausteine eingesetzt werden können. Notwendig ist folglich eine Spezifikation von Benutzerschnittstellen, die es ermöglicht wieder verwendbare Bausteine zu beschreiben, aus denen durch Komposition Benutzerschnittstellen generiert werden können.



Vanderdonckt beschreibt in [Van05] Anforderungen an wieder verwendbare Benutzerschnittstellen am Beispiel deklarativer Beschreibungen (für Benutzerschnittstellen). Er fasst die bisherigen Betrachtungen zusammen und ergänzt diese um weitere Aspekte:

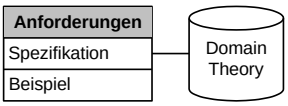
- Software Processability
  Eine Spezifikation muss präzise genug sein, um eine automatisierte Bearbeitung zu unterstützen, indem diese Spezifikation interpretiert oder direkt ausgeführt werden kann.
- Expressiveness
  Eine Spezifikation muss aussagekräftig genug sein, um Software-Entwicklungstechniken wie Ableitungen, Transformation und Abbildung von Modellen zu unterstützen.
- Standard Format
  Die Spezifikation sollte von möglichst vielen an dem Entwicklungsprozess beteiligten Partnern akzeptiert und auch benutzt werden.
- Human Readability
  Eine Spezifikation sollte von Menschen gelesen und verstanden werden können.
- Concision
  Die Spezifikation sollte kompakt genug sein, um sie zwischen unterschiedlichen Partnern austauschen zu können.

Vanderdonckt identifiziert dabei die „Software Processability“ und „Expressiveness“ als die wichtigsten Anforderungen. Die Erfüllung der anderen Kriterien sollte bei einer Auswahl hinter diesen beiden zurückstehen.

7.1.2 Anforderungen an die Dialogsteuerung

Aufgabe der Dialogsteuerung ist die Steuerung des Verhaltens der Präsentation. Als Steuereinheit bestimmt sie, welche Abläufe in einer Benutzerschnittstelle auftreten und welche Regeln dabei eingehalten werden müssen. Sie stellt sicher, dass Parameter aus Nutzerinteraktionen extrahiert und über die Anwendungsschnittstelle an die Anwendung weitergeleitet werden. Um dies zu gewährleisten, manipuliert die Dialogsteuerung die Präsentation auf Basis der in der Dialogsteuerung abgelegten Regeln und der Daten, die aus der Anwendungsschnittstelle entnommen werden können. Im Folgenden wird der Begriff Dialoglogik verwendet, um die Abläufe und Regeln, welche durch die Dialogsteuerung definiert werden, zu beschreiben.

Auch wenn die Dialogsteuerung auf Ebene von Benutzerinteraktionen, Dateneingaben des Benutzers und Daten aus der Anwendungsschnittstelle arbeitet und daher auf den ersten Blick nicht direkt plattformspezifische Eigenschaften benötigt, ist sie dennoch direkt an die Präsentation gekoppelt. Interaktionen des Benutzers über die Präsentation werden an die Dialogsteuerung weitergeleitet, so dass die Dialogsteuerung auf spezielle und oftmals auch plattformabhängige Ereignisse entsprechend reagieren muss. Betrachtet man die UIMS (User Interface Management Systeme), wie sie in Abschnitt 5.1.4.1 beschrieben wur-



den, so erfolgt die Spezifikation der gesamten Benutzerschnittstelle ausschließlich über das Dialogmodell (realisiert durch die Dialogsteuerung). Dies ist möglich, da das Dialogmodell Interaktionsmöglichkeiten des Benutzers modelliert und dadurch auch definiert, dass und oftmals auch welche Interaktionsobjekte in der Präsentation verfügbar sein müssen. Erweitert man die Dialogspezifikation um Constraints, wie beispielsweise im Dialognetz nach Janssen (vgl. [Jan93]), so lassen sich auch Informationen über das Verhalten der Präsentation modellieren.

Neben diesem plattformabhängigen Bereich beinhaltet die Dialogsteuerung auch plattformunabhängige Regeln, wie insbesondere im Zusammenspiel mit der Anwendungsschnittstelle deutlich wird. Die Anwendungsschnittstelle bietet Funktionen, die gewisse Parameter erwarten, um eine entsprechende Aufgabe erfüllen zu können. Das Wissen, dass bevor ein Ereigniss eintreten kann, zuerst entsprechende Parameter bereitstehen müssen, ist dabei offensichtlich plattformunabhängig.

Nichtsdestotrotz können alle Anforderungen, die an die Präsentation gestellt werden, auch auf die Dialogsteuerung übertragen werden. Sie benötigt auch eine Spezifikation, die über ein gewisses Abstraktionslevel verfügt, das den nach Vanderdonckt in [Van05] definierten Kriterien genügen sollte.

Weitere Anforderungen, die sich aus der Betrachtung der Schnittstelle der Dialogsteuerung zur Anwendungsschnittstelle ergeben, werden im folgenden Abschnitt genauer beschrieben.

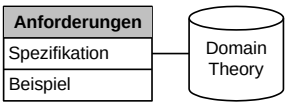
7.1.3 Anforderungen an die Anwendungsschnittstelle

Die Anwendungsschnittstelle bietet aus Sicht der Benutzerschnittstelle Daten, die von einer externen Quelle bereitgestellt werden. Die Anwendungsschnittstelle[34] kann diese Daten synchron (auf Anfrage der Benutzerschnittstelle) und asynchron (durch Ereignisse, die die Anwendungsschnittstelle an die Benutzerschnittstelle sendet) bereitstellen. Benutzereingaben wiederum können Parameter für die Anwendungsschnittstelle liefern.

Die Darstellung der Anwendungsschnittstelle als eine reine Datenquelle ist somit nicht ausreichend. Vielmehr verbirgt sich hinter dieser ein wesentlicher Teil der Anwendung, der selbst über eine Ausführungslogik und hoch komplexe Verarbeitung verfügt. Im Folgenden wird das Wissen um die Verarbeitung, die sich hinter der Anwendungsschnittstelle verbirgt, analog zur Dialoglogik „Anwendungslogik“ genannt. Auch wenn versucht wird, die Anwendungslogik möglichst von der Benutzerschnittstelle (und damit auch von der Dialoglogik) abzukoppeln, müssen beide in Beziehung zueinander gesetzt werden. Dynamische Daten, die für die Präsentation benötigt werden, müssen über die Anwendungsschnittstelle bezogen werden. Dies setzt aber voraus, dass alle Parameter, welche die Anwendungschnittstelle benötigt, um die angeforderten Daten bereit zu stellen, vorhanden sind. Das wiederum hat Implikationen für die Benutzerschnittstelle, die beispielsweise diese Parameter von dem Benutzer erfragen muss.

---

[34] Der Begriff der Schnittstelle wird in dieser Arbeit breiter interpretiert (Events sind ebenfalls Teil der Schnittstelle) und ist nicht auf die reine Bereitstellung von Funktionen zu reduzieren.



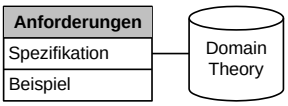
Legt man diese Sichtweise für die Beschreibung der Anwendungsschnittstelle zugrunde, so lassen sich Dialog- und Anwendungslogik durch unabhängige Abläufe modellieren, die sich nur an vorgegebenen Stellen punktuell synchronisieren müssen. Die Anwendungsschnittstelle lässt sich als eine Reihe von Diensten auffassen, die über einen Prozess miteinander verbunden sind. Im Laufe der Ausführung des Diensteprozesses können Daten von der Dialogsteuerung bereitgestellt werden, oder auch Daten an die Dialogsteuerung gesendet werden.

Diese Entkopplung beider Beschreibungen ist vorteilhaft, da somit sichergestellt werden kann, dass die Benutzerschnittstelle modifiziert und geändert werden kann, ohne den Diensteprozess anpassen zu müssen. Insbesondere wenn die Anwendung über unterschiedliche Endgeräte und Plattformen angeboten werden soll, oder wenn die Benutzerschnittstelle auf die speziellen Anforderungen eines Anwenders anzupassen ist, erweist sich diese Trennung als absolut notwendig.

Dass beide Abläufe (Dialog- und Anwendungslogik) aber nicht völlig entkoppelt werden können, liegt auf der Hand. Die Benutzerschnittstelle benötigt Daten aus Diensten, die dem Anwender präsentiert werden müssen. Andererseits benötigen auch Dienste Daten, die Nutzer durch ihre Interaktionen mit der Benutzerschnittstelle eingeben. Die Modellierung dieser Datenflüsse ist essenziell, da somit Punkte in beiden Abläufen identifiziert werden können, an denen eine Synchronisation erfolgen muss. Für diese Synchronisationspunkte müssen schließlich Schnittstellen definiert werden, die beide Abläufe voneinander entkoppeln.

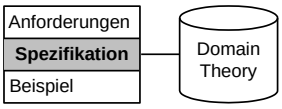
Die Beschreibung der Dialoglogik in Form von Abläufen setzt dabei nicht notwendigerweise ein ablauforientiertes Dialogmodell voraus. Auch in event-basierten Dialogspezifikationen, die über keine Ablaufdefinitionen verfügen, müssen Events als Synchronisationspunkte identifiziert werden. Zur Vereinfachung der folgenden Betrachtung wird aber ein ablauforientiertes Dialogmodell zugrunde gelegt.

Der in dieser Arbeit verwendete Ansatz zur Abbildung von Diensteprozessen auf Dialogabläufe basiert auf der Annahme, dass die Interaktionen der Benutzer einerseits Input liefern, der für die Ausführung von Diensten benötigt wird, und andererseits Diensteaufrufe durch die Interaktion des Anwenders gesteuert werden. Insbesondere die zweite Annahme impliziert eine Sichtweise, die sich von einer prozessorientierten Dienstebeschreibung grundsätzlich unterscheidet. Die prozessorientierte Dienstebeschreibung sieht Nutzerinteraktionen als Arbeitsschritte, die nicht vom System sondern von einer Person ausgeführt werden müssen. Diese Person führt den Arbeitsschritt aus und gibt die Kontrolle an eine Prozessmaschine weiter, die den weiteren Ablauf der Anwendung steuert. Die Interaktionsmodellierung hingegen impliziert eine Sichtweise, die den Anwender in den Vordergrund stellt und definiert die Interaktion des Anwenders als die steuernde Instanz. Anwender interagieren mit der Benutzerschnittstelle und lösen dadurch Diensteaufrufe aus, deren Ergebnisse dem Anwender angeboten werden. Der Anwender kann anschließend entscheiden, wie er mit den Ergebnissen fortfahren möchte.



Obwohl auf den ersten Blick beide Ansätze völlig unterschiedlich zu sein scheinen, stellen sie nur zwei unterschiedliche Sichtweisen auf denselben Sachverhalt dar. Während die Dienstebeschreibung Subprozesse und Funktionsfolgen in den Vordergrund stellt und beschreibt, wie Funktionen miteinander verknüpft werden können, beschreibt der Interaktionsablauf Nutzerinteraktionen und ihre Auswirkungen auf das Systemverhalten. Somit versteht sich die Interaktionsmodellierung als Erweiterung einer Dienstebeschreibung, die spezifiziert, wie Anwender mit dem System interagieren, um die für die Funktionen benötigten Parameter zu ermitteln.

Ein wesentlicher Vorteil dieser Sichtweise ist, dass bei der Interaktionsmodellierung von grundlegenden Problemstellungen bezüglich der Konzeption und Realisierung einer Anwendung abstrahiert werden kann. Problemstellungen wie der Komposition von Funktionen, die Transformation von Parametern, die Spezifikation von Funktionen etc., können bei der Erstellung der Benutzerschnittstelle vernachlässigt werden. Für die Interaktionsmodellierung müssen lediglich die Synchronisationspunkte zum Diensteprozess identifiziert und spezifiziert werden. Schließlich müssen Datensätze definiert werden, über welche der Informationsaustausch realisiert wird. Diese Datensätze sind nur eine Teilmenge der komplexen Datenmodelle, die gegebenenfalls in einer Dienstebeschreibung Verwendung finden.



7.2 Spezifikation von Benutzerschnittstellenfragmenten

Die Beschreibung der „Domain Theory“ als eine Menge von Benutzerschnittstellenfragmenten verlagert die Definition der Eigenschaften und auch einer passenden Spezifikation auf die Benutzerschnittstellenfragmente (kurz BSF).

In diesem Abschnitt wird basierend auf der in Abschnitt 7.1 eingeführten Sichtweise auf Benutzerschnittstellen und den damit einhergehenden Anforderungen definiert, was Benutzerschnittstellenfragmente als wieder verwendbare Bausteine einer Benutzerschnittstelle sind und wie diese spezifiziert werden können. Ein Benutzerschnittstellenfragment beschreibt einen wieder verwendbaren Teil eines Interaktionsablaufes und muss Wissen über die tatsächliche Realisierung der Benutzerschnittstelle beinhalten.

Demnach müssen zwei Zielsetzungen verfolgt werden: einerseits muss ein Benutzerschnittstellenfragment in der Lage sein, auf einer abstrakten Ebene die Kompositionen von Benutzerschnittstellen zu ermöglichen, andererseits muss es spezifisch genug sein, um daraus eine ausführbare Beschreibung generieren zu können.

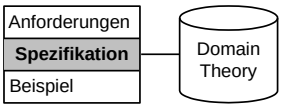
Basierend auf dieser Anforderung erfolgt die Konzeption einer Spezifikation. Weiterhin wird eine Event-basierte Dialogspezifikation definiert, welche die Grundlage für eine „Laufzeitumgebung“ von ausführbaren Benutzerschnittstellen liefert. Die Spezifikation von Benutzerschnittstellenfragmenten erfolgt erst am Ende dieses Abschnittes, nachdem alle notwendigen Bestandteile definiert wurden.

7.2.1 Konzeption von Benutzerschnittstellenfragmenten

Unter Berücksichtigung der vorhergehenden Betrachtungen und den Anforderungen an Benutzerschnittstellenfragmente wird für diese folgende Definition verwendet:

> *Ein Benutzerschnittstellenfragment (kurz BSF) realisiert einen abgrenzbaren und zusammengehörigen Ausschnitt aus einem Interaktionsablauf. Es stellt dabei einen eigenständigen Baustein dar, der einen wieder verwendbaren Teil einer Benutzerschnittstelle beschreibt. Die Beschreibung liegt dabei in einer Spezifikation vor, die eine automatische Generierung von ausführbaren Benutzerschnittstellen ermöglicht.*

Ein Interaktionsablauf beschreibt eine Folge von Interaktionen, die ein Anwender im Zuge der Durchführung einer Anwendung (bspw. „Mahlzeit planen“ beim Adipositas-Begleiter) mit der Benutzerschnittstelle ausführt. Ein Benutzerschnittstellenfragment ist als Teil eines Interaktionsablaufes definiert. Damit unterscheidet es sich von anderen Möglichkeiten zur Fragmentierung von Benutzerschnittstellen, in denen Benutzerschnittstellen nach funktionalen Aspekten (Präsentation, Dialog, Anwendungsschnittstelle) aufgeteilt werden, oder von GUI-Komponenten, die primär komplexe Interaktionsmöglichkeiten beschreiben. Interaktionsabläufe hingegen besitzen eine eigene graphische Repräsentation, beinhalten Interaktionsmöglichkeiten und beschreiben auch die Reaktion des Systems auf Interaktionen des Benutzers. Damit



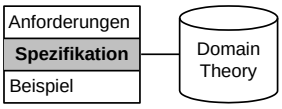
sind Benutzerschnittstellenfragmente Bausteine aus Sicht des Benutzers und beinhalten alle Aspekte, die Teil einer Benutzerschnittstelle sind.

Hervorgehoben werden insbesondere vier Anforderungen, denen Benutzerschnittstellenfragmente genügen müssen und die definieren, welche Teile von Interaktionsabläufen tatsächlich als Benutzerschnittstellenfragmente zu identifizieren sind:

- Abgrenzbarkeit
  Benutzerschnittstellenfragmente stellen Abläufe dar, die voneinander abgrenzbar sind. Insbesondere sollten sie vom Benutzer als Teilschritte identifiziert werden können.
- Zusammengehörigkeit
  Benutzerschnittstellenfragmente können mehrere Interaktionsschritte unterstützen. Die Schritte sollten dabei so gewählt werden, dass sie vom Benutzer als Ganzes wahrgenommen werden.
- Eigenständigkeit
  Jedes Benutzerschnittstellenfragment sollte einen wahrnehmbaren Ausschnitt beschreiben, der zwar auf vorhergegangenen Interaktionen beruhen kann, dennoch aber aus Sicht des Benutzers einen eigenen Zweck erfüllt.
- Wiederverwendbarkeit
  Ein Benutzerschnittstellenfragment sollte in der Lage sein, als Baustein zu fungieren, der in verschiedenen Interaktionsabläufen eingesetzt werden kann.

Schließlich müssen laut Definition Benutzerschnittstellenfragmente eine automatische Generierung von Benutzerschnittstellen ermöglichen. Dies setzt aber voraus, dass eine ausreichend detaillierte Spezifikation zur Beschreibung der Benutzerschnittstelle vorliegt. Zugrunde liegt die in Abschnitt 7.1 eingeführte Einteilung der Benutzerschnittstelle in die drei Bereiche Präsentation, Dialog und Anwendungsschnittstelle. Ein Benutzerschnittstellenfragment, das einen Teil des Interaktionsablaufs kapselt, wird folglich durch diese drei Bereiche spezifiziert.

Abbildung 33 veranschaulicht das Modell eines Benutzerschnittstellenfragmentes. Es wurde ein Kreis für die Beschreibung gewählt, um darzustellen, dass ein Benutzerschnittstellenfragment einen abgeschlossenen Teil der Benutzerschnittstelle beschreibt. Dabei werden die drei beschreibenden Teile (Präsentation, Dialog und Anwendungsschnittstelle) der Benutzerschnittstelle als gleichberechtigte und autonome Bereiche beschrieben, die zueinander über Schnittstellen (dargestellt durch einen kleinen weißen Kreis) in Beziehung stehen. Jeder Bereich hat definierte (nicht überschneidende) Schnittstellen zu den beiden angrenzenden Bereichen. Benutzerschnittstellenfragmente sind in sich abgeschlossen und ermöglichen die Generierung eines ausführbaren Teils einer Benutzerschnittstelle. Zusätzlich existiert eine Schnittstelle zu externen Diensten (Diensteschnittstelle), die als Teil der Anwendungsschnittstelle definiert wird. Diese Schnittstelle realisiert die Verbindung zu der Anwendungslogik, die von verschiedenen Dienstleistern erbracht werden kann.



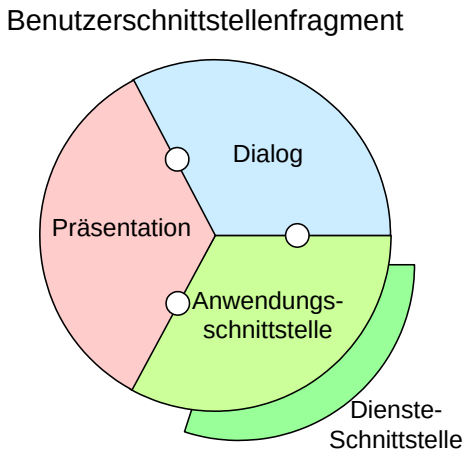


Abbildung 33: Teile eines Benutzerschnittstellenfragments

In den folgenden Abschnitten werden die einzelnen Bereiche eines Benutzerschnittstellenfragmentes eingehend beschrieben. Diese Beschreibung führt grundlegende Konzepte ein und bildet die Ausgangslage für eine formale Spezifikation, die in einem zweiten Schritt eingeführt wird.

Die Spezifikation wird exemplarisch an dem in in Abschnitt 5.6 (in Abbildung 26) eingeführten Szenario „Mahlzeit planen“ des Adipositas-Begleiters erläutert. Aus diesem Szenario wird ein Benutzerschnittstellenfragment ausgewählt, das einen Teil der dort definierten Interaktionen umsetzt.

7.2.1.1 Präsentation

Der Bereich Präsentation eines Benutzerschnittstellenfragmentes ist definiert durch eine Menge von voneinander unabhängigen Präsentationsfragmenten (kurz PF). Mit diesen Elementen wird die Präsentation eines Benutzerschnittstellenfragmentes beschrieben. Jedes PF beschreibt dabei die Darstellung eines Interaktionsschrittes, bestehend aus der Darstellung der Inhalte und der Interaktionsobjekte, welche die Interaktionsmöglichkeiten eines Benutzers beschreiben. Eine Interaktion (im Interaktionsablauf) ist dabei nicht unbedingt mit einer Eingabe oder einem Mausklick zu beschreiben. Auch eine Folge von Interaktionen, wie beispielsweise das Editieren von Text oder die Interaktion mit komplexen GUI-Objekten (geschachtelte Drop-Down Menüs oder graphische Manipulation von Interaktionsobjekten), sind Teil eines PF.

Jedes PF wird durch eine Spezifikation der Präsentation, der Interaktionsobjekte und durch Schnittstellen zu den beiden Bereichen Dialog und Anwendungsschnittstelle beschrieben. Ein PF agiert dabei als Black Box, das ausschließlich über seine Schnittstellen manipuliert wird. Die Spezifikation selbst wird erst nach dem Generierungsprozess dem Designer der Benutzerschnittstelle zur Verfügung gestellt und ermöglicht dann weitergehende Modifikationen. Zielsetzung der Schnittstellendefinition ist es, einerseits eine klare Trennung zu den anderen Bereichen (Dialog- und Anwendungsschnittstelle) zu definieren, und andererseits auch eine Komposition zu unterstützen.

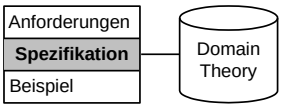


Abbildung 34 visualisiert ein PF mit seinen Schnittstellen zu den Bereichen Dialog und Anwendungsschnittstelle. Die gerichteten Kanten beschreiben Datenflüsse, wobei die Kantenrichtung die Richtung des Informationsflusses angibt. Die Schnittstellen ermöglichen dabei eine Abstraktion der internen Abläufe in einem Präsentationsfragment und erlauben es, unterschiedliche PF über einen Dialog zu einem Interaktionsablauf zu verbinden. Auch wenn PF sehr komplex werden können, bilden sie für die Modellierung einen atomaren Baustein, der nicht weiter zerlegt werden kann. Durch die fehlende Relation der PF untereinander erfolgt eine Verlagerung des Wissens über Abfolgen von Interaktionsschritten in die Dialogsteuerung. Das Wissen über die internen Interaktionsabläufe in einem PF bleibt in der Spezifikation gekapselt und entlastet die Dialogsteuerung somit von komplexen und hoch spezialisierten Ereignissen, die Interaktionsobjekte auslösen können.

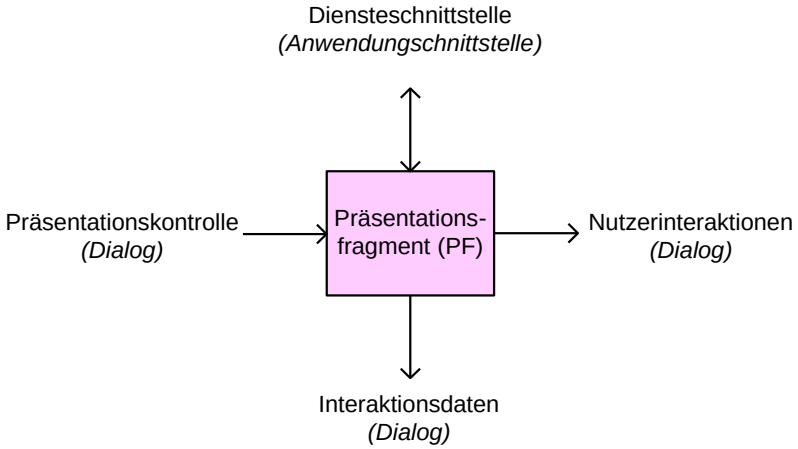


Abbildung 34: Schnittstellen eines Präsentationsfragments

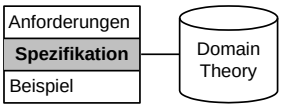
Im Folgenden werden die Schnittstellen (Präsentationskontrolle, Diensteschnittstelle, Interaktionsdaten und Nutzerinteraktion) und ihre Bedeutung genauer beschrieben.

Präsentationskontrolle

Jedes PF verfügt zum Ausführungszeitpunkt über einen Zustand, der definiert, ob ein PF sichtbar ist oder nicht. Ein PF kann somit zwei Zustände annehmen:

- Aktiviert
  Das PF ist für den Benutzer sichtbar und er kann jederzeit Interaktionen mit diesem durchführen. Nur aktivierte PF bieten auch Interaktionsmöglichkeiten an.
- Deaktiviert.
  Das PF ist für den Benutzer nicht sichtbar und damit auch nicht zugreifbar.

Über die Schnittstelle Präsentationskontrolle kann die Dialogsteuerung ein PF erzeugen, aktivieren oder deaktivieren, wobei durch die Menge aller aktivierten PF die Präsentation eines Benutzerschnittstellenfragmentes modelliert wird.



Neben den reinen Zustandszuweisungen, die von der Dialogsteuerung ausgelöst werden, können auch Daten übergeben werden, die beispielsweise Inhalte definieren, die über das PF präsentiert werden sollen. Welche Daten für die Darstellung benötigt werden, wird über diese Schnittstelle spezifiziert.

Weiterhin ermöglicht diese Schnittstelle die Steuerung von Interaktionsobjekten des PF. Jedes Interaktionsobjekt eines PF kann von der Dialogkontrolle aktiviert oder deaktiviert werden. Nur aktivierte Interaktionsobjekte können vom Benutzer manipuliert werden. Diese Schnittstelle wurde eingeführt, um der Dialogkontrolle die Synchronisation von Interaktionsabläufen zu ermöglichen, indem Interaktionsobjekte (beispielsweise Buttons) deaktiviert werden, bis ein bestimmtes Ereignis eintrifft, das diese wiederum aktiviert.

Diensteschnittstelle

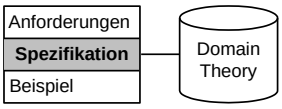
Die Diensteschnittstelle bietet dem PF den Zugriff auf externe Content-Quellen. Auch wenn über die Präsentationskontrolle Daten an das PF gesendet werden, sollten diese Daten möglichst nur Metainformationen enthalten. Das PF kann dann die Diensteschnittstelle nutzen, um zu den Metainformationen Inhalte zu erhalten, die schließlich dem Benutzer präsentiert werden. Dieses Vorgehen erlaubt eine weitgehende Entkopplung der Dialogkontrolle von präsentationsabhängigen Restriktionen.

Weiterhin können über die Diensteschnittstelle Datenflüsse von und zu komplexen Interaktionsobjekten erfolgen, die selbständig Daten aus externen Quellen beziehen und diese direkt auf der Benutzerschnittstelle darstellen. So könnten beispielsweise Änderungen des Datenbestandes direkt auf die Benutzerschnittstelle projiziert werden, ohne dass die Dialogkontrolle für eine Synchronisation dieser Abläufe verantwortlich wäre. Damit wird eine erhebliche Reduktion des Spezifikationsaufwandes der Dialogkontrolle erreicht und die Nutzung von komplexen Interaktionsobjekten ermöglicht, die außerhalb der verwendeten Spezifikation liegen.

Nutzerinteraktionen

Die Schnittstelle Nutzerinteraktionen beschreibt Benachrichtigungen (im Folgenden Events genannt) der Dialogsteuerung, die durch Interaktion des Benutzers mit Interaktionsobjekten ausgelöst werden. Welche Events in welcher Reihenfolge erfolgen, ist vom Verhalten des Benutzers abhängig. Zu unterscheiden sind dabei direkte und indirekte Events, die unterschiedliche Auswirkungen auf die Komposition haben können.

Indirekte Events beschreiben Interaktionen, denen andere Interaktionen des Benutzers vorhergegangen sind, welche Interaktionsdaten beschreiben. Ein typisches Beispiel für einen indirekten Event ist das Abschicken eines Formulars, das vorher vom Benutzer ausgefüllt wurde. Direkte Events beschreiben hingegen Interaktionen, die erst durch ihr Auftreten Interaktionsdaten liefern. So steht beispielsweise bei einem PF, das eine Liste aus Verweisen realisiert, erst bei der Auswahl eines Verweises auch eine Information über die getroffene Auswahl zur Verfügung.



Diese Unterscheidung ist notwendig, um Aussagen über die Kompositionsmöglichkeiten von zwei PF zu treffen. In Abschnitt 8.4.5 wird beispielsweise eine Kompositionsart eingeführt, die gleichzeitig Daten aus zwei PF erfassen muss. Dieses ist nur möglich, wenn beide PF Formulare darstellen, die parallel ausgefüllt und gleichzeitig abgeschickt werden. Bei zwei Link-Listen ist dies aber nicht möglich, da der Benutzer nicht gleichzeitig zwei Verweise auswählen kann.

Sowohl direkte als auch indirekte Events können Daten enthalten, die durch Interaktionen des Benutzers mit den PF generiert wurden.

Interaktionsdaten

Neben den Events bietet die Schnittstelle „Interaktionsdaten“ eine weitere Möglichkeit, Informationen über Benutzerinteraktionen aus einem PF zu gewinnen. Über diese Schnittstellen kann die Dialogsteuerung Daten auslesen, die der Benutzer durch Interaktionen mit einem PF festgelegt hat, ohne ein Event auszulösen. Ein Beispiel für eine solche Art von Interaktion ist das Editieren eines Textes. Über diese Schnittstelle kann jederzeit der aktuelle Text aus dem PF ausgelesen werden.

Diese letzte Schnittstelle mag redundant erscheinen, da alle Daten prinzipiell auch mit Hilfe von Events verschickt werden können, wie es in formularbasierten Web-Anwendungen umgesetzt wird. Diese Schnittstelle bietet aber die Möglichkeit, unabhängig von Interaktionen mit dem PF (beispielsweise ausgelöst durch externe Ereignisse oder Interaktionen eines anderen PF) Daten aus der Schnittstelle aus zulesen.

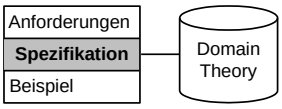
Spezifikation eines PF

Die Spezifikation des PF wird an einem Benutzerschnittstellenfragment veranschaulicht, welches einen Teil der Interaktionen für das Szenario „Mahlzeit planen“ (vgl. Abbildung 26 in Abschnitt 5.6) des Adipositas-Begleiters realisiert.

Spezifiziert wird das Benutzerschnittstellenfragment „Rezepte suchen in Kategorien mit Filtern“, welches die Interaktion „Rezept suchen“ (vgl. Abbildung 26 in Abschnitt 5.6) realisiert. Angeboten wird in einem ersten Schritt eine Liste aus Kategorien („Mahlzeit-Typ-Liste“ genannt). Über diese Liste können Sub-Kategorien erreicht, oder aber zur übergeordneten Kategorie zurückgekehrt werden. Jeder Kategorie werden 0 bis n Elemente zugeordnet, die in einer separaten Liste („Rezept-Liste“ genannt) präsentiert werden. Über ein Stichwort in der „Rezept-Liste“, das durch den Benutzer eingegeben wird, können die Resultate gefiltert werden.

Aus dieser Beschreibung ist ersichtlich, dass das Benutzerschnittstellenfragment durch zwei Präsentationsfragmente („Mahlzeit-Typ-Liste“ und „Rezept-Liste“) beschrieben wird.

Im Folgenden wird die Spezifikation der beiden Präsentationsfragmente vorgestellt:



Mahlzeit-Typ-Liste

***Name**: Mahlzeit-Typ-Liste*

***Spezifikation**: Mahlzeit-Typ-Liste.xaml*

***Nutzerinteraktionen**:*

{ „Kategorie ausgewählt“, indirekt, Kategorie}
{ „Eine Ebene hoch“, direkt, Kategorie}

***Präsentationskontrolle**:*

NULL → erzeugen → NULL
NULL → deaktivieren → NULL
({Kategorie}, {KategorieListe}) → aktivieren → NULL
({Kategorie}, {KategorieListe})→ ändern → NULL

Events → IOaktivieren → NULL // Aktivieren des Interaktionsobjektes, welches durch das „Event“ identifiziert wird.
Events → IOdeaktivieren → NULL //Deaktivieren des Interaktionsobjektes, welches durch das „Event“ identifiziert wird.

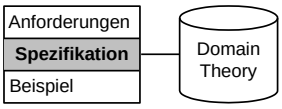
***Interaktionsdaten**:*

NULL → aktuelleKategorie → {Kategorie}

***Status**: deaktiviert*

Jedes PF verfügt über einen eindeutigen Namen (im Beispiel „Mahlzeit-Typ-Liste“), über den das PF identifiziert werden kann und unter dem es für die anderen Bereiche (Dialog und Anwendungsschnittstelle) adressierbar ist. Die Spezifikation des PF, also Informationen über Layout, das interne Verhalten und wie das PF auf dem Endgerät zu rendern ist, wird über ein externes Dokument (im Beispiel Mahlzeit-Typ-Liste.xaml) definiert. Das Kürzel .xaml repräsentiert eine Spezifikation auf Basis von XAML (vgl. Abschnitt 2.4.3) und wurde bewusst gewählt, um auf die notwendige Mächtigkeit der Spezifikation zu verweisen. Welche Spezifikation tatsächlich verwendet wird, ist für das Benutzerschnittstellenfragment-Modell nicht weiter relevant.

Das PF kann zwei Events (im Beispiel „Kategorie ausgewählt“, „Eine Ebene hoch“) auslösen. Ein Event wird durch einen eindeutigen Namen, den Typ und einen Datensatz (im Beispiel „Kategorie“) definiert. Jedes dieser Events ist dabei eindeutig einem Interaktionsobjekt des PF zugeordnet. Diese Eigenschaft macht man sich bei der Präsentationskontrolle zu nutze, indem zu aktivierende bzw. deaktivierende (IOaktivieren und IOdeaktivieren) Interaktionsobjekte durch die Events identifiziert werden. Für die Beschreibung von Schnittstellen wird dabei folgende Schreibweise benutzt: <Eingaben> → <Schnittstellenname> → <Ausgaben>. Weitere Schnittstellen des PF sind erzeugen, ändern, aktivieren und deaktivieren



des PF selbst, wobei für aktivieren und ändern optional ein Datensatz benötigt wird. Im Beispiel definiert der Datensatz, welche Kategorie angezeigt werden soll, und schließlich die Liste der dazugehörigen „Unterkategorien“. Weiterhin kann über die Schnittstelle der Interaktionsdaten „aktuelleKategorie“ die aktuell präsentierte Kategorie abgefragt werden.

Das zweite Präsentationsfragment „Rezept-Liste“ ist analog dazu wie folgt anzugeben:

Rezept-Liste

***Name**: Rezept-Liste*

***Spezifikation**: Rezept-Liste.xaml*

***Nutzerinteraktionen**:*

{ „Rezept ausgewählt“, direkt, Rezept}
{ „Seite-vor“, direkt, SeitenNR}
{ „Seite-zurück“, direkt, SeitenNR}
{ „Suchbegriff angegeben“, indirekt, Stichwort}

***Präsentationskontrolle**:*

NULL → erzeugen → NULL
NULL → deaktivieren → NULL
({Kategorie}, {RezeptListe}) → aktivieren → NULL
({Kategorie}, {RezeptListe})→ ändern → NULL
({SeitenNR}, {RezeptListe})→ ändern → NULL
({Stickwort}, RezeptListe})→ ändern → NULL

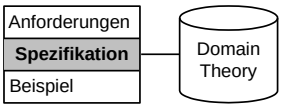
Events → IOaktivieren → NULL // Aktivieren des Interaktionsobjektes, welches durch das „Event“ identifiziert wird.
Events → IOdeaktivieren → NULL //Deaktivieren des Interaktionsobjektes, welches durch das „Event“ identifiziert wird.

***Interaktionsdaten**:*

NULL→ aktuellesStichwort → {Stichwort}
NULL→ aktuelleSeite→ {SeitenNR}
NULL→ aktuelleKategorie → {Kategorie}

***Status**: deaktiviert*

Das Präsentationsfragment „Rezept-Liste“ verfügt im Vergleich zur „Mahlzeit-Typ-Liste“ über eine Reihe zusätzlicher Interaktionsmöglichkeiten („Rezept auswählen“, „Seite vor“, „Seite zurück“ und „Suchbegriff angeben“). Diese Interaktionsmöglichkeiten spiegeln sich in der Präsentationskontrolle und auch in



den Interaktionsdaten des Präsentationsfragmentes wider. So wird neben der Kategorie, durch welche die Liste aller Rezepte definiert wird, auch eine Seitenzahl und auch ein Stichwort angegeben, mit Hilfe dessen die Rezeptliste nochmals gefiltert werden kann.

7.2.1.2 Dialogsteuerung

Der Bereich Dialogsteuerung eines Benutzerschnittstellenfragmentes ist definiert durch eine Menge von Dialogfragmenten (kurz DF). Mit diesen Element wird die Dialogsteuerung eines Benutzerschnittstellenfragmentes beschrieben. Jedes DF spezifiziert die Reaktion des Benutzerschnittstellenfragmentes auf Ereignisse, die von der Präsentation, der Anwendungsschnittstelle aber auch von anderen DF ausgelöst werden können. Es existiert eine eindeutige Zuordnung von Ereignissen zu DF. Jedes DF reagiert auf genau ein Ereignis und besitzt eine eigene Verarbeitungslogik, welche die Aktivitäten des DF beschreibt, die beim Eintreten eines Ereignisses durchgeführt werden. Die Beschreibung der Verarbeitungslogik liegt in einer Pseudo-Programmiersprache vor, die in unterschiedlichen Implementierungen interpretiert werden kann.

Abbildung 35 visualisiert die Schnittstellen eines DF und nutzt die Beschreibungselemente, die zur Beschreibung der Präsentationsfragmente (vgl. Abbildung 34) eingeführt wurden.

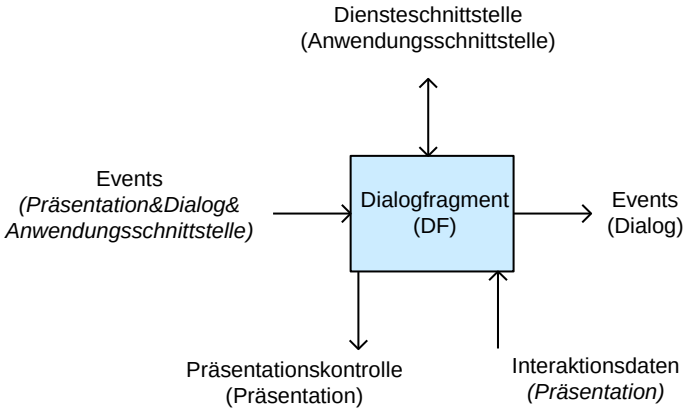
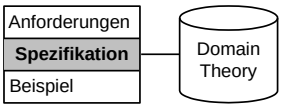


Abbildung 35: Schnittstellen eines Dialogfragments

Die Events, durch die jedes DF definiert ist, haben eine eindeutige Semantik:

- Events aus der Präsentation beschreiben Nutzerinteraktionen, die durch Interaktionsobjekte ausgelöst wurden.
- Events aus der Anwendungsschnittstelle beschreiben Benachrichtigungen aus externen Systemen. Dieser Mechanismus ermöglicht die Integration von Dienstleistungen, die aktiv Informationen an einen Benutzer zustellen.
- Events aus Dialogen beschreiben die Delegation der Verarbeitungslogik. Diese sind insbesondere für die Komposition von Benutzerschnittstellenfragmenten relevant, da damit eine Übergabe des Interaktionsflusses erfolgen kann.



Die Verarbeitungslogik eines DF wird nur aktiv, wenn ein entsprechendes Ereignis eingetreten ist. Jedes Ereignis kann dabei auch Inhalte für die Verarbeitung liefern. Ein DF kann für die Verarbeitung auf folgende Schnittstellen zugreifen:

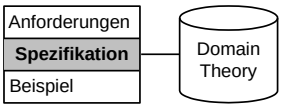
- Präsentationskontrolle
  Wie in Abschnitt 7.2.1.1 beschrieben, kann ein DF den Status eines PF manipulieren oder Interaktionsobjekte des PF aktivieren oder deaktivieren. Mit diesem Mechanismus können DF typische Dialogabläufe realisieren, indem sie auf Interaktionen des Benutzers neue PF aktivieren bzw. deaktivieren. Für die Beschreibung des Interaktionsablaufes wird ein Event-basierter Ansatz verwendet.
- Daten aus der Interaktion
  Wie in Abschnitt 7.2.1.1 beschrieben, kann ein DF über diese Schnittstelle Daten aus einem PF auslesen, ohne dass hierzu ein Event des PF ausgelöst wurde.
- Events (Dialog)
  Ein DF kann selbst Ereignisse auslösen, die von anderen DF bearbeitet werden.
- Schnittstelle für dynamische Daten
  Über diese Schnittstelle kann ein DF auf Daten von externen Diensten zugreifen. Grundsätzlich sollte ein DF in seiner Verarbeitungslogik primär Interaktionsabläufe beschreiben und möglichst nicht für die Transformation von Daten in passende Datenstrukturen verwendet werden. Diese Aufgabe ist an die Diensteschnittstelle zu delegieren.

Betrachtet man die Definition der Dialogsteuerung und die Schnittstellen, über welche die einzelnen DF verfügen, so wurde implizit eine Event-basierte Beschreibung des Dialogablaufes zugrunde gelegt. Diese Wahl ist durch die Forderung nach einer Beschreibung von Benutzerschnittstellenfragmenten, die nahe an einer ausführbaren Spezifikation liegt, begründet. Dementsprechend können DF als Event-Handler aufgefasst werden, die für die Verarbeitung eines speziellen Ereignisses ausgelegt sind. Analog dazu kann die Präsentation, aber auch die Anwendungsschnittstelle als Event-Quelle definiert werden, und stellt somit eine direkte Beziehung zu der Dialogsteuerung her.

Spezifikation

Wie bereits bei der Spezifikation des Präsentationsfragmentes wird auch die Spezifikation des Dialogfragmentes anhand eines Beispiels veranschaulicht. Das Benutzerschnittstellenfragment „Rezepte suchen in Kategorien mit Filtern“ aus dem Szenario „Mahlzeit planen“ (vgl. Abbildung 26 in Abschnitt 5.6) des Adipositas-Begleiters, das auch für die Spezifikation des Präsentationsfragmentes herangezogen wurde, wird an dieser Stelle weiter entwickelt.

Legt man die im vorherigen Abschnitt 7.2.1.1 definierten Präsentationsfragmente zu Grunde, sind Dialogfragmente für jede der angegebenen „Nutzerinteraktionen“ zu spezifizieren, da jede der Nutzerinteraktionen ein „Event“ auslöst, das durch ein eigenes Dialogfragment behandelt werden muss. Im Einzelnen sind dies:



- { „Kategorie ausgewählt“, indirekt, Kategorie} // PF: „Mahlzeit-Typ-Liste“
- { „Eine Ebene hoch“, direkt, Kategorie} // PF: „Mahlzeit-Typ-Liste“
- { „Rezept ausgewählt“, direkt, Rezept} // PF:„Rezept-Liste“
- { „Seite-vor“, direkt, SeitenNR} // PF: „Rezept-Liste“
- { „Seite-zurück“, direkt, SeitenNR} // PF: „Rezept-Liste“
- { „Suchbegriff angegeben“, indirekt, Stichwort} // PF: „Rezept-Liste“

Anhand der beiden DF „Kategorie ausgewählt“ und „Seite-vor“ wird die Spezifikation erläutert.

Der folgende Ausschnitt veranschaulicht die Spezifikation des DF „KategorieAusgewählen“, welches auf eine Benutzerinteraktion reagiert. Das DF nimmt das Event „Kategorie ausgewählt“ entgegen, das von einem PF ausgelöst wird, und aktiviert die beiden PF „Mahlzeit-Typ-Liste“ und „Rezept-Liste“ (falls Rezepte für die Kategorie verfügbar sind), wie sie im vorhergehenden Abschnitt 7.2.1.1 beschrieben wurden.

KategorieAuswählen

***Name**: KategorieAuswählen*

***Event**: {„Kategorie ausgewählt“}*

Konfiguration

INTEGER: rezepteProSeite = 10

Variablen

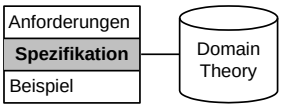
{Kategorie}: kategorie
INTEGER: seite = 1
{RezeptListe}: rezeptListe
{KategorieListe}: kategorieListe

Statements

(1) kategorie = „Kategorie ausgewählt“getEventData() // Abfragen der Kategorie aus dem Event

(2) rezeptListe = ASF-Rezeptbuch.getRezepte(kategorie, rezepteProSeite, Seite) // Aus der Anwendungsschnittstelle (vgl. 7.2.1.3) wird die Liste der Rezepte ausgelesen.

(3) kategorieListe = ASF-Rezeptbuch.getSubKategorien(kategorie) // Aus der Anwendungsschnittstelle (vgl. 7.2.1.3) wird die Liste der Kategorien ausgelesen.



(4) Mahlzeit-Typ-Liste.erzeugen() // Erzeugen des PF
Mahlzeit-Typ-Liste.aktivieren(kategorie, kategorieListe) // aktivieren des PF mit den entsprechenden Parametern

(5) IF rezeptListe <> NULL THEN

Rezept-Liste.erzeugen(); // Erzeugen des PF.
Rezept-Liste.aktivieren(kategorie, rezeptListe); // Nur wenn auch Rezepte zu einer Kategorie verfügbar sind, soll auch die Rezeptliste angezeigt werden.
Rezept-Liste.IOdeaktivieren(Seite-zurück) // Kategorie Auswählen definiert immer die erste Seite, daher wird das Interaktionsobjekt (Seite-zurück) deaktiviert.
IF ASF-Rezeptbuch.getAnzahlSeiten(kategorie, rezepteProSeite)<=1 THEN Rezept-Liste.IOdeaktivieren(Seite-vor) // Wenn nur eine Seite verfügbar ist, wird das Interaktionsobjekt (Seite-vor) deaktiviert.

Jedes DF verfügt über einen Namen als eindeutiger Bezeichner und ein Event, das spezifiziert, wann das DF mit der Verarbeitung seiner Statements beginnt. Über die Konfiguration ist es möglich, das Verhalten von DF zu beeinflussen. In dem Beispiel kann festgelegt werden, wie viele Produkte auf einer Seite angezeigt werden, was durch den Konfigurationsparameter „ProduktepProSeite“ definiert wird. Weiterhin verfügt ein DF über eine Reihe von Variablen, die es für die Verarbeitung seiner Statements einsetzen kann.

Die Statements beschreiben schließlich die Verarbeitung des Events in Form einer Pseudo-Programmiersprache. Die einzelnen Direktiven werden sequentiell abgearbeitet und können die beschriebenen Schnittstellen nutzen.

Das Dialogfragment „Seite-vor“ ist analog zu „KategorieAuswählen“ definiert. Es ist dabei sicher gestellt, dass auch tatsächlich eine weitere Seite existiert, da sonst das Event nicht hätte auftreten können. Zu beachten ist weiterhin, dass, falls ein „Stichwort“ für die Filterung der Rezepte gesetzt wurde, dieses ebenfalls mit berücksichtigt werden muss.

Seite-vor

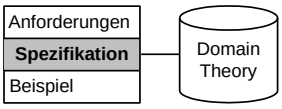
***Name**: Seite-vor*

***Event**: {„Seite-vor“}*

Konfiguration

Integer: RezepteProSeite = 10

Variablen



{Kategorie}: kategorie
INTEGER: seitenr
{RezeptListe}: rezeptListe
{Stichwort}: stichwort

Statements

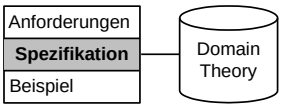
(1) stichwort = Rezept-Liste.aktuellesStichwort()
(2) seitennr= Rezept-Liste.aktuelleSeite()
(3) kategorie = Rezept-Liste.aktuelleKategorie()

(4) rezeptListe = ASF-Rezeptbuch.getRezepte (kategorie, stichwort, ProdukteProSeite, seitennr+1) // Ermitteln der neuen Rezeptliste für die nachfolgende Seite.
(5) Rezept-Liste.ändern(seitennr+1, rezeptListe); // Ändern des PF mit den neuen Daten.
IF seitennr=1 THEN Rezept-Liste.IOaktivieren(Seite-zurück) // Da dieses Event nur von einer existierenden Vorgängerseite aufgerufen werden kann, wird das Interaktionsobjekt (Seite-zurück) aktiviert.
IF ASF-Rezeptbuch.getAnzahlSeiten(kategorie,stichwort, RezepteProSeite)= seitennr+1 THEN Rezept-Liste.IOdeaktivieren(Seite-vor) // Wenn keine nachfolgende Seite existiert, wird das Interaktionsobjekt (Seite-vor) deaktiviert.

7.2.1.3 Anwendungsschnittstelle

In Analogie zur Präsentation und Dialogsteuerung erfolgt auch die Beschreibung des Bereiches der Anwendungsschnittstelle über ein Anwendungsschnittstellenfragment (kurz ASF). Dieses erfüllt zwei Aufgaben: einerseits stellt es eine Schnittstelle zu externen Diensten und Anwendungen bereit, über die Daten von und zu PF und DF fließen können, andererseits realisiert es selbst eine Datenverarbeitungslogik, auf die DF und PF zurückgreifen können. Damit bietet der Bereich der Anwendungsschnittstelle weit mehr als einen reinen Wrapper, der Daten aus externen Quellen für die Dialogverarbeitung aufbereitet. Dabei ist zu berücksichtigen, dass ein ASF sehr wohl Daten zwischenspeichern, transformieren und auch komplexe Operationen durchführen kann.

Abbildung 36 veranschaulicht die Schnittstellen eines ASF. So bietet es spezielle Schnittstellen für DF und PF an und ist weiterhin in der Lage, Events zu versenden, für deren Verarbeitung DF bereitstehen. Weiterhin definiert das ASF die einzige Schnittstelle zu externen Diensten und Datenquellen. Diese Schnittstelle definiert, welche Daten von dem Benutzerschnittstellenfragment benötigt werden. Hierfür stellt jedes Benutzerschnittstellenfragment eine Schnittstellenspezifikation bereit, die von den Diensten implementiert werden muss.



Realisiert werden kann solch eine Schnittstelle beispielsweise über WSDL (Web Service Definition Language (vgl. [W3C07])), aber auch andere Schnittstellendefinitionen können hier verwendet werden. Zusätzlich kann das ASF selbst Schnittstellen anbieten, die es externen Diensten ermöglichen, Nachrichten an Benutzerschnittstellenfragmente zu senden und damit aktiv Aktionen auf der Benutzerschnittstelle auszulösen.

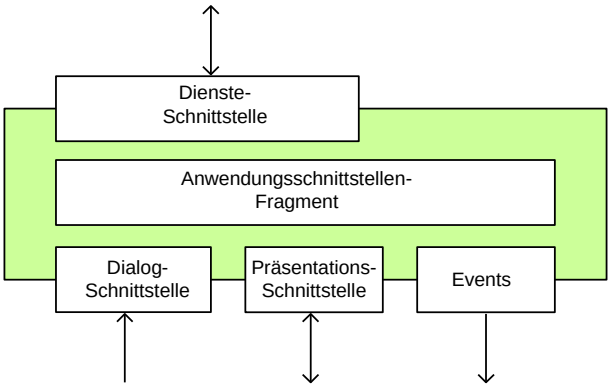
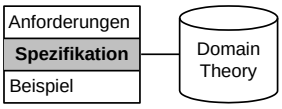


Abbildung 36: Detaillierung der Anwendungsschnittstelle

Das ASF wird für die Komposition von Benutzerschnittstellen als Black-Box angesehen, welche lediglich über ihre Schnittstellen definiert wird. Eine Realisierung des ASF kann in einer beliebigen Form erfolgen und ist ausschließlich von der Ausführungsumgebung abhängig. Erst zum Ausführungszeitpunkt müssen tatsächlich Daten vorliegen, für den Kompositionsprozess ist ein ASF ausreichend durch seine Schnittstellen definiert.

Für die Realisierung der beiden Dialogfragmente „Kategorie ausgewählt“ und „Seite-vor“ sind die folgenden Schnittstellen bereitzustellen:

- *{Kategorie}->ASF-Rezeptbuch.getSubKategorien->{KategorieListe}*
  Liefert eine Liste aller Subkategorien, basierend auf der übergebenen Kategorie.
- *({Kategorie} ,{Stichwort}, Integer) -> ASF-Rezeptbuch.getAnzahlSeiten -> Integer*
  Liefert die Anzahl der Seiten (Rezeptseiten) basierend auf einer Kategorie und gegebenenfalls einem Stichwor,. wobei durch den Integer-Wert die maximale Anzahl von Rezepten auf einer Seite festlegt wird.
- *({Kategorie}, {Stichwort}, Integer, Integer) -> ASF-Rezeptbuch.getRezepte -> {RezeptListe}*
  Liefert eine Liste von Rezepten basierend auf einer Kategorie und gegebenenfalls einem Stichwort, wobei der erste Integer-Wert die maximale Anzahl von Rezepten auf einer Seite festleg, und der zweite die Seite definiert, für welche die Rezeptliste erstellt wird.



7.2.1.4 Datenmodell für Benutzerschnittstellenfragmente

Der Datenaustausch zwischen Präsentations-, Dialog- und Anwendungsschnittstelle wird auf Basis von Datenstrukturen definiert. Auch wenn die interne Verarbeitung der Daten in diesen Datenstrukturen über Schnittstellen gekapselt werden kann, derer sich Dialogfragmente bedienen, ist für die Komposition von Benutzerschnittstellenfragmenten eine genauere Definition notwendig. Hier müssen Datenflüsse analysiert werden, um Eingabedaten für das komponierte Benutzerschnittstellenfragment zu bestimmen, oder um zu entscheiden, an welcher Stelle im Dialogablauf bestimmte Daten zur Verfügung stehen. Dies erfordert aber Analysen und die Definition von Transformationsregeln auf diesen Datenstrukturen, um schließlich entscheiden zu können, wann und wie ein Datenaustausch erfolgen kann.

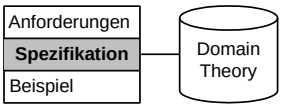
Für die Definition von Datenstrukturen kann auf eine Vielzahl von Schemadefinitionen zurückgegriffen werden. Zurzeit werden DTD (Dokumenttypdefinition vgl. [W3C06]) oder aktuell XML-Schema (vgl. [W3C04]) für die Beschreibung von Datenstrukturen in XML eingesetzt. Der Vergleich und die Integration von Schema-Definitionen und schließlich die automatische Ableitung von Transformationsregeln ist ein vieldiskutiertes Thema im Umfeld von Web-Service Komposition und Datawarehouse Anwendungen. Rahm und Bernstein stellen in [RB01] die Problemstellungen des Schema-Matching vor und präsentieren existierende Ansätze und Werkzeuge für diese Problemstellung. Ersichtlich ist aus den verwendeten Ansätzen, dass eine allgemeine Lösung nicht trivial ist und vielfach auch auf menschliche Nachbereitung angewiesen ist.

In dieser Arbeit können Restriktionen und Vorbedingungen definiert werden, um die Komplexität der Problemstellung zu reduzieren. Dies ist möglich, da die Modellierung der Benutzerschnittstellenfragmente domänenspezifisch in einem geschlossenen System erfolgt, und nur eine definierte Personengruppe für die Erstellung der Beschreibungen verantwortlich ist. Schließlich übersteigt die Komplexität moderner Schema-Definitionen bei weitem die Anforderungen, die ein Datenaustausch in einem geschlossenen System mit sich bringt.

Es wird also eine Basis für die Definition von Datenstrukturen benötigt, die einfach analysiert und für die Transformationsregeln zwischen Datenstrukturen automatisch ermittelt werden können. Eine Grundlage einer einfachen Definition von Datenstrukturen bietet Lewerentz in [Lew00]. Sie nutzt diese, um Daten, die in Folge von Interaktionsprozessen anfallen, zu modellieren, aber auch für die weitere Verarbeitung dieser Daten in Funktionen. Definiert ist diese Struktur durch folgende Regeln:

<data>	<name>: <d_type> | NULL
<name>	<word>
<d_type>	<basic> | <complex>>
<d_list>	<data> | <data> , <d_list>
<basic>	STRING | CHAR | NAT | REAL | BOOL | DATE
<complex>	SET (<data>) | RECORD (<d_list>) | <data>

Tabelle 9: Erzeugungsregeln für Datenstrukturen (in Anlehnung an [Lew00])



Die Definition von Datenstrukturen nach Lewerentz basiert auf einfachen Konstruktionsregeln. Ein Datum hat dabei einen Namen als eindeutigen Bezeichner und ist entweder einfach, dann ist er ein grundlegender Datentyp (String, Char, Nat, Real, Bool oder Date) oder er ist komplex. Komplexe Daten definieren eine hierarchische Struktur:

- SET
  eine Liste bestehend aus Elementen derselben Datenstruktur.
- RECORD
  Eine endliche Menge von Datenstrukturen.
- <data>
  Diese Struktur beschreibt die Möglichkeit der Referenzierung einer existierenden Datenstruktur.

Letzteres Konstrukt beschreibt eine Erweiterung der Definition nach Lewerentz und bietet die Möglichkeit, komplexe Strukturen durch Komposition zu beschreiben. Diese Eigenschaft spielt für die Komposition von Benutzerschnittstellen eine Rolle, wenn Datenflüsse in Benutzerschnittstellen analysiert werden müssen, birgt aber auch die Möglichkeit, rekursive Strukturen zu definieren.

Die folgende Datenstruktur veranschaulicht den Einsatz des Schemas am Beispiel der Stammdaten eines Patienten. Ein Patient ist demnach durch seine Basisdaten (PatientBasisDaten), seine Adresse und seine Diagnosen definiert, wobei ein Patient beliebig viele Diagnosen haben kann.

Patient: ***RECORD***

PatientBasisDaten: ***RECORD***

Vorname: STRING

Nachname: STRING

Geschlecht: M|F

Geburtsdatum: DATE

Versicherungsnummer: STRING

Adresse: ***RECORD***

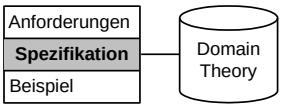
Straße: STRING

PLZ: STRING

Stadt: STRING

*Diagnosen:****SET***

Diagnose: Diagnose



7.2.2 Spezifikation von Benutzerschnittstellenfragmenten

Die Spezifikation von Benutzerschnittstellenfragmenten muss zwei Anforderungen genügen: einerseits muss sie eine Grundlage für die Beschreibung der Komposition von Benutzerschnittstellenfragmenten schaffen, andererseits aber auch eine Dialogspezifikation liefern, aus der ausführbare Benutzerschnittstellen generiert werden können. Hierzu wird eine semiformale Definition von Präsentations-, Dialog- und Anwendungsschnittstellenfragmenten eingeführt. Weiterhin werden Methoden und Konstrukte spezifiziert, die für die Ausführung von Benutzerschnittstellenfragmenten benötigt werden.

Grundlage für die Spezifikation sind die vorhergehenden Betrachtungen und Beschreibungen der drei Bereiche eines Benutzerschnittstellenfragmentes, denen ein Event-basiertes Dialogsystem zugrunde liegt.

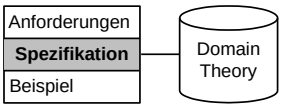
In einem ersten Schritt wird hierzu eine Event-basierte Dialogspezifikation eingeführt, die durch Fragmente aus den Bereichen Präsentation, Dialog und Anwendungsschnittstelle definiert wird. Anschließend wird kurz umrissen, wie aus solch einer Dialogspezifikation eine ausführbare Benutzerschnittstelle realisiert werden kann. Basierend auf der Dialogspezifikation wird im Abschluss ein Benutzerschnittstellenfragment definiert.

7.2.2.1 Eine Event-basierte Dialogspezifikation

Ausgangslage für die Beschreibung von Benutzerschnittstellen ist ein vereinfachtes Event-Modell. Das Event-Model abstrahiert dabei von herkömmlichen Programmiersprachen und bietet eine Grundlage für eine semi-formale Spezifikation. Zugrunde gelegt wird ein Event-Modell, das auf Event-Hierarchien oder Delegates verzichtet, wie sie in höheren Programmiersprachen wie JAVA oder C# verwendet werden. Das Event-Modell besteht aus einer Menge von Events, die von unterschiedlichen Quellen (im Folgenden Event-Quelle genannt) versendet werden. Jedem Event sind ein oder mehrere Event-Handler zugeordnet, die ausgelöst durch das Event eine definierte Verarbeitungsroutine durchführen. Auf dieser Grundlage ist eine Dialogspezifikation folgendermaßen definiert:

> *Eine Event-basierte Dialogspezifikation E-DS ist ein 7-Tupel E-DS = {P, D, AS, Events, StartDF, DataSet, SessionData} mit:*
> *- P als eine Menge von PF, wie in Abschnitt 7.2.1.1 beschrieben;*
> *- D als eine Menge von DF, wie in Abschnitt 7.2.1.2 beschrieben;*
> *- AS als eine Menge von ASFs, wie in Abschnitt 7.2.1.3 beschrieben;*
> *- Events ist eine Menge von Ereignissen (Events genannt);*
> *- StartDF ist eine DF, das den Dialog initiiert und startet;*
> *- DataSet als eine Menge von Datenstrukturen, wie in Abschnitt 7.2.1.4 beschrieben;*
> *- SessionData ist eine spezielle Datenstruktur für den Austausch von Daten zwischen DF.*

Im Folgenden werden die einzelnen Elemente eines E-DS genauer spezifiziert.



DataSet

DataSet beschreibt eine endliche Menge von Datenstrukturen, wie sie in Abschnitt 7.2.1.4 beschrieben wurden. DataSet enthält alle Datenstrukturen, die in einer Benutzerschnittstellendomäne verwendet werden, und ist für die Beschreibung von Interaktionen aller Anwendungen einer Domäne gleich. Damit definiert die Menge der Datenstrukturen in DataSet nicht nur einfache Daten einer Anwendung, sondern beschreibt vielmehr ein einheitliches Datenmodell für unterschiedlichste Anwendungen einer Domäne. Zu berücksichtigen ist dabei, dass DataSet nur Datensätze beinhaltet, die für die Beschreibung von Benutzerschnittstellen benötigt werden, und somit nur einen stark eingeschränkten Teil eines globalen Modells bildet.

Events

Events beschreibt die Menge aller Events einer Dialogspezifikation. Welche Events in einer Dialogspezifikation verwendet werden ist abhängig von den verwendeten DF, PF und ASFs. Jedem Event ist mindestens ein DF zugeordnet, das für die Verarbeitung des Events verantwortlich ist.

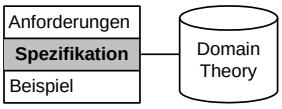
> *Ein Event E wird definiert als 4-Tupel E = {eventName, typ, eventQuelle, eventData}, wobei gilt: eventName ∈ EventName und EventName ist eine Menge von Bezeichnern für E. Das Objekt typ ist Element der Menge {direkt, indirekt} und bezeichnet die in Abschnitt 7.2.1.1 eingeführten direkten oder indirekten Event-Typen. EventQuelle beschreibt den Auslöser des Events, das PF, DF aber auch das ASF sein können. EventData ∈ DataSet ist schließlich eine Datenstruktur, welche die Daten beschreibt, die dem Event von der Event-Quelle mitgegeben werden können.*

Events-auslösende Komponenten sind so genannte *Event-Quellen*. Dieses können PF, DF und ASFs sein, die durch eine Menge von Events definiert werden, die diese Komponenten auslösen können. Eine Event-Quelle definiert somit lediglich, welche Events ausgelöst werden können. Wann diese Events ausgelöst werden, hängt von der auslösenden Komponente ab. So sendet ein PF Events aufgrund von Nutzerinteraktionen, während ein ASF Events aufgrund von externen Ereignissen verschickt. DF lösen schließlich Events aus, die eine Übergabe der Dialogkontrolle an ein anderes DF realisieren.

Präsentationsfragmente P

P beschreibt eine Menge von PF, wie sie in Abschnitt 7.2.1.1 beschrieben wurden.

> *Ein PF ist definiert als 7-Tupel PF={name, Spezifikation, Nutzerinteraktionen, Präsentationskontrolle-Schnittstellen, Interaktionsdaten-Schnittstelle, Anwendungs-Schnittstellen, Status}, wobei name ∈ PFName und PFName eine Menge von Bezeichnern für PF ist. Spezifikation ist ein Platzhalter für eine Spezifikation des PF, das die graphische Repräsentation, Interaktionsobjekte und gegebenenfalls auch das Verhalten von Interaktionsobjekten beschreibt. Nutzerinteraktionen ⊂ Events ist eine Menge von Events, die ein PF auslösen kann. Alle Events eines PF werden dabei*



durch Nutzerinteraktionen ausgelöst. Eine Schnittstelle wird adressiert durch einen Schnittstellennamen und ist eine Relation Interface: $DataSet^1$ x $DataSet^2$ x ...x $DataSet^n$ → $DataSet^m$. Und $DataSet^1$, ...,$DataSet^n$,... ,$DataSet^m$ ∈ DataSet.

Die Interaktionsdaten-Schnittstelle ist eine Schnittstelle: NULL → getInteractionData → Data, die keine Parameter als Übergabewert erwartet und einen Datensatz Data ∈ DataSet zurückliefert. Wie in Abschnitt 7.2.1.1 beschrieben, enthält der Datensatz Informationen aus allen vorhergehenden Interaktionen, die mit dem PF durchgeführt wurden, und ermöglicht es DF, auf diese Daten zuzugreifen. Die Anwendungsschnittstelle definiert Schnittstellen, auf welche das PF zugreift, um Daten von externen Diensten zu erhalten. Status ist ein Element der Menge {„aktiviert", „deaktiviert"}. Die Präsentationskontrolle-Schnittstelle enthält die in Abschnitt 7.2.1.1 definierten Schnittstellen für eine Steuerung des PF durch DF.

Anzumerken ist, dass die Anwendungsschnittstelle lediglich definiert, welche Schnittstellen vom PF verwendet werden. Implementiert werden diese Schnittstellen in einem ASF.

Dialogfragmente D

D beschreibt die Menge von DF, wie sie in Abschnitt 7.2.1.2 beschrieben wurden.

Nur DF realisieren Event-Handler, die auf genau ein Event reagieren. Ein DF ist demnach wie folgt definiert:

Ein DF ist ein 5-Tupel DF={DFName, event, Anwendungs-Schnittstellen, Var, Statement} für das gilt:

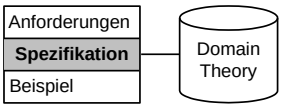
(1) DFName ∈ DialogfragmentName und DialogfragmentName ist eine Menge von Bezeichnern für Dialogfragmente.

(2) Anwendungs-Schnittstellen definieren eine Menge von Schnittstellen zu ASFs, auf die das DF im Laufe seiner Verarbeitung des Events zugreifen kann.

(3) event ∈ Event definiert das Event, auf welches das Dialogfragment reagiert.

(4) Var ist eine Menge von Variablen, definiert als grundlegenden Datentypen wie Integer, Bool, Real, Char oder Sting. Aber auch komplexe Datenstrukturen, wie sie als DS (Datenstrukturen) definiert wurden, können in Var enthalten sein.

(5) Statement ist eine Sequenz von Elementen des Typus Direktive, wobei fünf Arten von Direktiven möglich sind:



- <var> = <Ausdruck>

Diese Direktive berechnet einen neuen Wert für eine Variable. Ein Ausdruck kann dabei aus Variablen, Konstanten und Operatoren bestehen.

- IF <condition> THEN <statement> ELSE <statement>

Diese Direktive beschreibt die bedingte Ausführung von zwei Direktiven. Die Bedingung kann aus Variablen, Konstanten und logischen Operatoren bestehen.

- While <condition> Do <statement>

Diese Direktive beschreibt die wiederholte Ausführung von Direktiven. Die Direktive wird solange ausgeführt, bis die Bedingung nicht mehr dem logischen Wert wahr entspricht.

- <var> = <Interface><(var^1,var^2,...,var^n)>

Diese Direktive beschreibt den Aufruf einer Schnittstelle auf entweder einem PF oder einem ASF mit den Aufrufparametern var^1,var^2,...,var^n. Schnittstellen liefern immer einen Rückgabewert, der aber auch NULL sein kann. Anzumerken ist, dass var, var^1, var^2,...,var^n Elemente aus DS sind.

- sendEvent(Event)

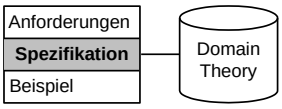
Diese Direktive versendet ein Event, das von einem dazu passenden Event-Handler (ein Dialogfragment) weiter verarbeitet wird.

Ein Dialogfragment ist im Wesentlichen durch seine Direktiven definiert, die beschreiben, wie auf bestimmte Events reagiert werden soll. Dabei wird über das Event definiert, welche Dialogfragmente für die Verarbeitung verantwortlich sind. Sollten mehrere Dialogfragmente für ein und dasselbe Event verantwortlich sein, so wird jedes Dialogfragment für eine Verarbeitung angestoßen. Über die Reihenfolge der Ausführung werden keine Aussagen getroffen.

Anwendungsschnittstelle AS

Die Anwendungsschnittstelle AS definiert und implementiert Schnittstellen für PF und DF und ermöglicht es, Events zu versenden, wie in Abschnitt 7.2.1.3 beschrieben wurde. Weiterhin wird durch AS auch eine Schnittstelle für externe Dienste und Anwendungen definiert.

Eine Anwendungsschnittstelle ist definiert als AS = {InterneSchnittstellen, Externe-Schnittstellen, Events}. InterneSchnittstellen beschreiben dabei die Menge aller Schnittstellen, die von PF und DF aufgerufen werden, die ExterneSchnittstellen die Menge aller Schnittstellen für externe Dienste. Weiterhin verfügen AS über eine Menge von Events, die sie selbst auslösen kann.



SessionData

SessionData ist eine spezielle Datenstruktur mit definierten Schnittstellen, die es ermöglicht, Daten zwischen PF auszutauschen. Jedes PF muss Daten, die für andere PF von Interesse sein könnten, in SessionData hinterlegen.

> *SessionData sind definiert als eine Menge von Datensätzen, die Daten aus Benutzerinteraktionen enthalten. Jeder Datensatz besteht dabei aus einem eindeutigen Namen und einer Datenstruktur, wie sie in DataSet definiert wurde. SessionData ermöglicht den Zugriff auf Interaktionsdaten aus allen Dialogfragmenten und ist insbesondere für die Komposition von Benutzerschnittstellen relevant, wenn Benutzerdaten aus einem Benutzerschnittstellenfragment an ein anderes übergeben werden sollen. SessionData verfügt über drei Schnittstellen, die den Zugriff auf die hinterlegten Datensätze einschränken:*
> *(1) Data → SessionData.put→NULL,*
> *(2) Name → SessionData.get→Struktur*
> *(3) Name → SessionData.delete→NULL*
>
> *Data ist ein 2-Tupel (Name, Struktur) für das gilt: Name ist ein eindeutiger Bezeichner für Datensätze und Struktur eine Datenstruktur aus DataSet.*

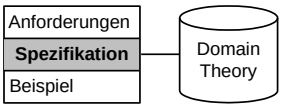
StartDF

StartDF ist ein ausgezeichnetes DF aus der Menge aller DF aus D, welches den Beginn des Interaktionsablaufes kennzeichnet. Dieses DF wird aufgerufen und ausgeführt, wenn das BSF gestartet wird. Bevor dieses DF nicht ausgeführt wurde, ist die Benutzerschnittstelle nicht aktiv und reagiert auf keine anderen Events, die beispielsweise durch die Anwendungsschnittstelle ausgelöst werden können.

7.2.2.2 Ausführung einer Event-basierten Dialogspezifikation

Die im vorhergehenden Abschnitt 7.2.2.1 eingeführte Dialogspezifikation ist als solche noch nicht ausführbar, und benötigt noch weitergehende Informationen (über Endgeräte, parallele Fenster, Frames) und eine Laufzeitumgebung, welche die Spezifikation interpretiert. Schließlich muss auch eine Infrastruktur geschaffen werden, welche die Event-Verwaltung, das Mapping der Schnittstellenspezifikation auf konkrete Funktionen und schließlich auch das Rendering der Spezifikation der PF realisiert.

In diesem Abschnitt wird die Ausführung einer Dialogspezifikation kurz umrissen und dargestellt, wie die einzelnen Objekte der Spezifikation interagieren, um Interaktionsabläufe zu beschreiben. Insbesondere wird auf die Realisierung von dynamischen Interaktionsabläufen eingegangen, die abhängig vom Nutzerverhalten unterschiedliche Instanzen eines Präsentationsfragmentes realisieren. Aber auch andere grundlegende Mechanismen werden kurz vorgestellt, die beispielsweise über Direktiven in DF realisiert werden und damit nicht direkt Teil der Dialogspezifikation sind.



Dialogstart

Grundlage für die Ausführung ist eine Dialogspezifikation, wie sie in Abschnitt 7.2.2.1 eingeführt wurde. Im initialen Zustand ist jedes Präsentations- und Dialogfragment und die Anwendungsschnittstelle in Form von einer Instanz verfügbar, die durch den jeweiligen Namen identifiziert werden kann. Alle möglichen Events, die gegebenenfalls über die Anwendungsschnittstelle an das Dialogsystem versendet werden, werden ignoriert. Einzig das Event, welches das StartDF (im Folgenden auch das Startdialogfragment genannt) auslöst, wird weitergeleitet. Jedes PF hat initial den Zustand „deaktiviert“ und SessionData ist eine leere Datenmenge.

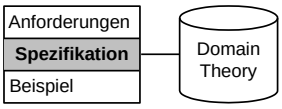
Die Ausführung startet mit dem Aufruf des Startdialogfragmentes, eines ausgezeichneten Dialogfragmentes, das durch ein Event ausgelöst wird. Dieses Event kann auch Informationen beinhalten, die in der Event-Verarbeitung berücksichtigt werden. Die Event-Verarbeitung kann auf die definierten Direktiven und die Schnittstellen der AS zurückgreifen.

Eine besondere Bedeutung kommt der Aktivierung der PF zu. Solange alle PF den Status deaktiviert haben, sind keine Benutzerinteraktionen möglich, da keine Präsentation verfügbar ist, mit der der Benutzer interagieren kann.

Aktivierung eines PF

Ein PF kann nur durch ein DF aktiviert werden, indem das DF die Direktive „<PF-Name>.aktivieren(Data)“ aufruft. Data beschreibt dabei eine Datenstruktur aus DataSet, welche Informationen enthält, die für die Generierung des PF benötigt werden. Dies können Daten sein, die über das PF präsentiert werden sollen, aber auch Konfigurationen, welche die Darstellung betreffen. Das Aktivieren eines PF ist gleichbedeutend mit der Generierung einer graphischen Repräsentation des PF. Dabei definiert ein PF noch nicht, ob es in einem eigenen Fenster, in einem Frame oder zusammen mit anderen PF in einer Maske oder einem Formular sichtbar ist. Die Laufzeitumgebung ist somit einerseits für die Generierung der Repräsentation zuständig, andererseits aber auch für die Integration des PF in die Benutzerschnittstelle. Dabei bedient sich die Laufzeitumgebung primär der Spezifikation des PF. Hier ist hinterlegt, wie eine graphische Repräsentation aufgebaut werden muss, aber auch, wie dynamische Daten zu integrieren sind, die beispielsweise aus der Übergabe (Data) ermittelt werden, oder auch über die Anwendungsschnittstelle bezogen werden. Der Anwendungsschnittstelle kommt dabei eine zusätzliche Bedeutung zu. Diese liefert nicht nur reine Daten, sondern vielmehr bereits aufbereitete Teile der Benutzerschnittstelle.

Obwohl die Dialogspezifikation noch keine Aussagen über die genaue Ausgestaltung von Benutzerschnittstellen trifft, da hier noch nicht definiert wird, wie das PF genau präsentiert werden soll, ist durch die Menge der sichtbaren PF eindeutig definiert, welche Teile der Benutzerschnittstelle sichtbar sind. Damit wird auch bestimmt, welche Interaktionsmöglichkeiten zur Verfügung stehen.



Benutzerinteraktionen

Die Benutzerinteraktion erfolgt über Interaktionsobjekte der graphischen Repräsentation. Hier müssen Benutzerinteraktionen, die innerhalb eines PF erfolgen, und Interaktionen, die ein Event auslösen, unterschieden werden. Erstere haben keine Auswirkung auf die Dialogspezifikation, sondern beschreiben einen Teil der Interaktion, die über die Spezifikation in dem PF gekapselt ist. Solche Interaktionen können einfache Navigation in hierarchischen Menüstrukturen beschreiben, aber auch die Interaktion mit komplexen Interaktionsobjekten, wie beispielsweise die Auswahl eines Datums mit Hilfe eines Kalenders. Solch ein Kalender kann als graphisches Objekt realisiert werden, das beispielsweise unterschiedliche Ansichten unterstützt. Gewöhnlich werden solche Objekte als Softwarekomponenten realisiert, die eine gewisse Funktionalität verbunden mit einer Benutzerschnittstelle anbieten. Der Verzicht auf solche Komponenten und die Realisierung der notwendigen Interaktionen über die Dialogspezifikation würde deren Komplexität vervielfachen. Die Laufzeitumgebung ist dafür verantwortlich, Interaktionen des Benutzers mit diesen Softwarekomponenten zu koordinieren.

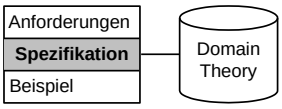
Interaktionen, die über die Dialogspezifikation verarbeitet werden, versenden definierte Events. Für das Versenden der Events ist wiederum die Laufzeitumgebung verantwortlich. Wobei in der Spezifikation der PF definiert wird, was für Events bei bestimmten Benutzerinteraktionen ausgelöst werden. Welche Events ein PF auslösen kann, ist Teil der Definition eines PF. Dadurch wird eine Analyse ermöglicht, ohne die Spezifikation genauer betrachten zu müssen. Die Dialogspezifikation muss dabei sicherstellen, dass für jedes mögliche Event auch ein DF zur Verfügung steht, das dieses Event verarbeiten kann. Weiterhin existiert eine eindeutige Zuordnung von Events zu Interaktionsobjekten des PF. Diese Beziehung ermöglicht das Aktivieren und Deaktivieren von Interaktionsobjekten durch die definierte Schnittstelle der PF.

Das Event-verarbeitende DF führt nun die definierten Direktiven aus, die seine Verarbeitung beschreiben. Dabei können neue PF aktiviert oder deaktiviert werden, Informationen aus der SessionData ausgelesen oder hinzugefügt werden, oder neue Events verschickt werden, die andere DF aktivieren. Die Laufzeitumgebung ist für die Interpretation und Ausführung der Direktiven verantwortlich.

Dynamische PF

Von jedem PF existiert im initialen Zustand der Dialogspezifikation genau eine Instanz. In Benutzerschnittstellen ist aber die Erzeugung von unterschiedlichen Instanzen desselben Oberflächenobjektes durchaus üblich. So kann beispielsweise ein Benutzer aus einer Liste heraus gleichzeitig mehrere Fenster öffnen[35], die ihm Informationen zu unterschiedlichen Elementen anbieten. Jedes dieser Fenster ist bis auf die dargestellten Inhalte identisch. Da nicht vorhersehbar ist, wie viele Fenster geöffnet werden, müssen Mechanismen geschaffen werden, die das dynamische Erzeugen von Fenstern ermöglichen.

[35] Im Web-Browsern wird dies oftmals über sogenannte Pop-Up Fenster realisiert.



PF bieten hierzu die Schnittstelle „erzeugen“ an. Im Gegensatz zur „aktivieren“-Schnittstelle wird nicht der Status eines Fensters geändert, sondern tatsächlich ein neues PF erzeugt. Problematisch gestaltet sich in der vorliegenden Dialogspezifikation die Adressierung des erzeugten PF, da ausschließlich bereits im Initialzustand definierte Bezeichner verwendet werden. Ein dynamisch erzeugtes Fenster kann aber nicht mehr von einem DF adressiert werden, da entweder alle dynamisch generierten PF den gleichen oder unterschiedliche Namen erhalten. Um dennoch dynamische PF zu realisieren, wird folgendermaßen vorgegangen:

Alle dynamisch erzeugten PF werden in der Dialogspezifikation mit gleichem Bezeichner gekennzeichnet. Einzig die Laufzeitumgebung verwaltet die unterschiedlichen Instanzen. Jede dieser dynamisch erzeugten Instanzen ist in der Lage, die in der Spezifikation definierten Events zu versenden. Die Laufzeitumgebung generiert zusätzlich für jedes Event eines dynamischen PF einen eigenen Namen und sendet diesen mit jedem Event mit (vgl. EventQuelle in der Eventdefinition). Ein DF ist somit nicht in der Lage, zu unterscheiden, ob nun das Event von einer dynamisch generierten Instanz stammt oder nicht. Dialogfragmente, die nun eine dedizierte Instanz manipulieren sollen, verwenden nicht den Namen des PF zur Adressierung der Schnittstelle, sondern den Parameter EventQuelle in dem Event. Damit ist sichergestellt, dass nur das PF manipuliert wird, das auch Auslöser des Events ist. Wird der Name des PF zur Adressierung verwendet, so werden durch die Laufzeitumgebung alle Instanzen des PF manipuliert.

Dieser Mechanismus ermöglicht es dem Entwickler der Dialogspezifikation, dynamische PF zu generieren und zu kontrollieren, überträgt ihm aber auch die Verantwortung dafür, die unterschiedlichen Instanzen zu verwalten.

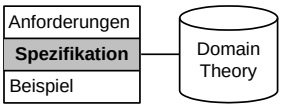
Dialogende

Die Ausführung einer Dialogspezifikation gilt als beendet, wenn kein PF mehr den Status aktiviert hat und die Ausführung aller DF abgeschossen ist. Anzumerken ist, dass in diesem Zustand immer noch Events aus der Anwendungsschnittstelle ausgelöst werden könnten. Diese werden aber nicht verarbeitet, es sei denn das Startdialogfragment wird ausgelöst.

Plattformabhängige Basisinteraktionen

Benutzerschnittstellen zeichnen sich durch eine gewisse Menge an Basisfunktionalitäten aus, die Benutzer erwarten vorzufinden. Abhängig von der Art der Benutzerschnittstelle (fensterbasierte Systeme, sprachgesteuerte Systeme, Masken) können solche Interaktionen das Minimieren, Schließen oder Verschieben von Fenstern sein, oder die Zurück- oder Vorwärts-Buttons in Web-Browsern.

Da diese Funktionalitäten und auch die Einbettung in die Benutzerschnittstelle aber abhängig von der Art der Benutzerschnittstelle und damit plattformabhängig sind, sind sie nicht Teil der allgemeinen Dialogspezifikation. Ähnlich wie die Interaktion mit komplexen Interaktionsobjekten eines PF, die nicht in der Dialogspezifikation, sondern in der Spezifikation des PF abgelegt sind und von der Laufzeitumgebung ausgeführt werden, sind auch diese Basisinteraktionen nicht Teil der Dialogspezifikation.



7.2.2.3 Spezifikation eines Benutzerschnittstellenfragmentes

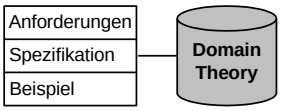
Die Definition eines Benutzerschnittstellenfragmentes ist größtenteils bereits durch die Dialogspezifikatition gegeben, die sich bereits Fragmente aus den Bereichen Präsentation, Dialog und Anwendungsschnittstelle bedient. Formal wird ein Benutzerschnittstellenfragment definiert als:

> *Ein Benutzerschnittstellenfragment ist ein 7-Tupel BF={P, D, ASF, StartData, StartDF, SubDataSet, SubEvents}, wobei P, D, ASF und StartDF wie in Abschnitt 7.2.2.1 definiert sind. StartData ist eine Datenstruktur wie sie für SessionData definiert wurde. SubDataSet ist eine Teilmenge von DataSet und SubEvents eine Teilmenge von Events.*

Ein Benutzerschnittstellenfragment besteht somit aus einer Menge von Präsentations- und Dialogfragmenten, verfügt über eine eigene Anwendungsschnittstelle und besitzt ein Startdialogfragment, das den Beginn des Interaktionsablaufes bestimmt. Weiterhin definiert ein Benutzerschnittstellenfragment Datenstrukturen und Events, die bei der Dialogausführung benötigt werden. Benutzerschnittstellenfragmente können somit als Teilmenge einer Dialogspezifikation aufgefasst werden, die zwar bereits ausführbar ist, die aber nur einen Teil der notwendigen PF, DF und AS für die Ausführung liefert.

Zusätzlich definiert ein Benutzerschnittstellenfragment die Datenstruktur StartData, welche Datensätze bestimmt, die für die Ausführung benötigt, aber nicht in dem Benutzerschnittstellenfragment erhoben werden. Somit wird über StartData eine Schnittstelle zu vorhergehenden Benutzerschnittstellenfragmenten definiert, indem eine gewisse Datenmenge gefordert wird, um ein Benutzerschnittstellenfragment auszuführen. Die Analogie zwischen StartData und SessionData ist bewusst gewählt, da die geforderten Daten für die Ausführung in die Datenstruktur SessionData überführt werden müssen.

Die in diesem Kapitel eingeführte Spezifikation erlaubt eine implementierungsnahe Beschreibung von Benutzerschnittstellenfragmenten. Weiterhin wurde gezeigt, wie diese zu nutzen ist, um ausführbare Benutzerschnittstellen durch diese Fragmente zu realisieren. Die Fragestellung, auf welche Weise Benutzerschnittstellenfragmente komponiert werden, wurde nur indirekt durch die Spezifikation von Benutzerschnittstellenfragmenten beantwortet. Erst in Kapitel 8 werden Werkzeuge und Methoden zur Komposition vorgestellt.



7.3 Realisierung der „Domain Theory"

Benutzerschnittstellenfragmente stellen die wieder verwendbaren Bausteine dar, die für das „Compositional Modeling" benötigt werden. Sie beschreiben auf einer sehr implementierungsnahen Ebene Teile einer Benutzerschnittstelle, aus denen durch Komposition dieser Elemente neue Benutzerschnittstellen generiert werden können.

Analog zu dem Objektmetaphern-Katalog (vgl. Abschnitt 5.4) müssen auch die Benutzerschnittstellenfragmente auf eine Anwendungsdomäne ausgerichtet werden. Ziel ist es, einen „Baukasten" (im Umfeld des „Compositional Modeling" „Domain Theory" genannt) zu definieren, mit dem ausführbare Benutzerschnittstellen für diese Anwendungsdomäne generiert werden können.

Welchen Umfang die „Domain Theory" haben muss, um eine ausreichende Wissensbasis bereit zu stellen, ist abhängig von den verwendeten Objekt- und Interaktionsmetaphern und damit vom Objektmetaphern-Katalog und vom verwendeten Nutzungskontextmodell. Letzteres beschreibt die verwendeten Endgeräte, Fähigkeiten der Benutzer und das Umfeld der Nutzung. Aus diesen Komponenten lassen sich Anforderungen definieren, denen die Benutzerschnittstellenfragmente der „Domain Theory" genügen müssen.

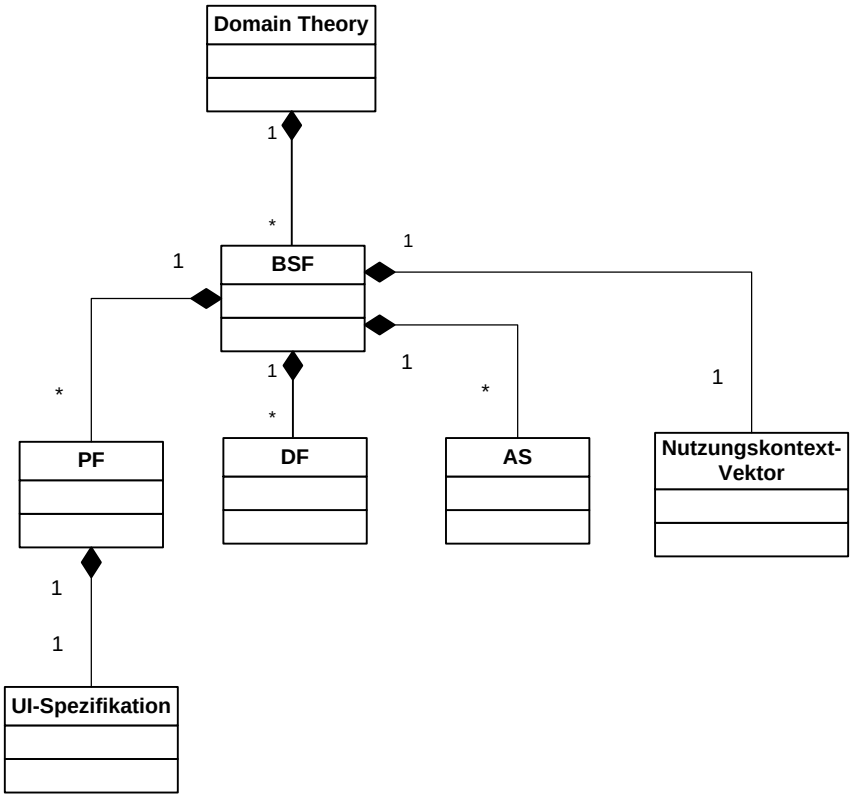


Abbildung 37: Struktur der „Domain Theory"

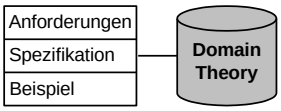


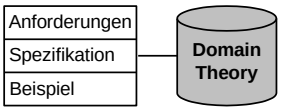
Abbildung 37 gibt den Aufbau der „Domain Theory“ wieder. Definiert wird sie als eine Menge von Benutzerschnittstellenfragmenten wie sie in Abschnitt 7.2.1 beschrieben wurden. Zusätzlich wird jedem Benutzerschnittstellenfragment ein „Nutzungskontextvektor“ zugeordnet, der Informationen über Eigenschaften des Benutzerschnittstellenfragments beinhaltet. Beschrieben werden diese Eigenschaften über die Elemente des Nutzungsprofils (vgl. Abschnitt 6.3) mit Ausnahme des UAProf[36] Bestandteiles, da ein BSF möglichst geräteunabhängig mit Hilfe einer XML-basierten Spezifikationssprache beschrieben wird. Auf eine Beschreibung des Nutzungskontextvektors (kurz auch Kontextvektor genannt) wird an dieser Stelle verzichtet und auf Abschnitt 8.2.3 verwiesen. Dort wird erläutert, wie der Nutzungskontextvektor aufgebaut ist und wie er benutzt werden kann, um die Eignung eines Benutzerschnittstellenfragmentes in Bezug auf ein gegebenes Nutzungsprofil zu errechnen.

Die „Domain Theory“ beschreibt in erster Linie „technische Bausteine“, die über den Objektmetaphern-Katalog erschlossen werden. Auf eine Kategorisierung und Beschreibung der Elemente innerhalb der „Domain Theory“ wird verzichtet, da diese indirekt über den Objektmetaphern-Katalog gegeben sind (vgl. Abschnitt 5.4).

Tabelle 10 veranschaulicht einen Ausschnitt aus der „Domain Theory“ für den Adipositas-Begleiter, um darzustellen, welche Benutzerschnittstellenfragmente eine „Domain Theory“ definieren können. Für die Beschreibung der Benutzerschnittstellenfragmente wurden „semantische“ Informationen über die Benutzerschnittstellenfragmente herangezogen, die nicht Teil der „Domain Theory“ sind, aber das Verständnis für den Einsatz der Benutzerschnittstellenfragmente erleichtern.

	Name	**Beschreibung**	**#PF**	**#DF**	**Nutzungskontextvektor** (informale Beschreibung)
Primitive BSF	Verweisliste	Auswahl eines Elementes aus einer Liste	1	1	Geeignet für alle Nutzungsprofile. Die Parameter des Nutzungskontextvektors sind sehr niedrig „bewertet“, da andere BSF (sollte es diese geben) zu bevorzugen sind.
	Checkboxen	Auswahl eines oder mehrerer Elemente einer Liste	1	1	
	Formularfeld	Eingabe in ein Freitextfeld	1	1	
	Freitexteingabe	Eingabe eines Freitextes (Textfeld)	1	1	
	Text	Ausgabe von Text	1	1	
	Bild	Ausgabe eines Bildes	1	1	
	...				
Domänenunabhängige BSF	Kategorieliste	Realisiert eine Verweisliste mit einer zusätzlichen Interaktionsmöglichkeit, die es erlaubt, eine Ebene in der Kategorie zurück zu springen.	1	2	- Alle Geräte - keine Einschränkungen - keine Einschränkungen
	Uhrzeit ändern	Graphische Darstellung einer Uhr (Zifernblatt) mit der Option Stunden- und Minutenzeiger zu manipulieren.	1	1	- PC und PDA (Maus, Touch-Display erforderlich) - Erfordert feinmotorische Fähigkeiten, effiziente Durchführung

[36] UAProf wird im Nutzungsprofil zur Beschreibung der Eigenschaften des verwendeten Endgerätes benötigt.

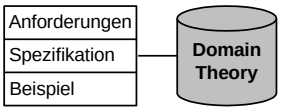


					- Gut für mobile Szenarien geeignet
	Nach Stichworten suchen (einfach)	Möglichkeit ein Stichwort als Suchparameter zu definieren. Suchergebnis ist eine Liste, aus der ein Element ausgewählt werden kann.	2	3	- Alle Geräte - erfordert Texteingabe, einfache sequentielle Durchführung - keine Einschränkungen
	In Kategorien suchen (mit Filtern)	Angeboten wird eine Liste aus Kategorien. Über diese Liste können Sub-Kategorien erreicht, oder aber zur übergeordneten Kategorie zurückgekehrt werden. Jeder Kategorie können 0 bis n Elemente zugeordnet werden, die in einer separaten Liste präsentiert werden.	2	5	- Alle Geräte (2 PF müssen gleichzeitig darstellbar sein) - Einfache Bedienung (keine Texteingabe), aber paralleler Ablauf - Für mobile Szenarien geeignet
	Katalogsuche (komplex)	Angeboten wird eine Suche, die über Stichworte und gleichzeitig über Kategorien realisiert wird.	5	8	- Nicht für Set-Top-Boxen (3 PF müssen gleichzeitig darstellbar sein) - komplexe, effiziente Durchführung - keine Einschränkungen
	...				
Domänenspezifische BSF	Mahlzeit betrachten (einfach)	Realisiert die sequentielle Darstellung einer Mahlzeit bestehen aus dem Rezept (Zutaten, Zubereitung, ...), des Mahlzeittyps (bspw. Mittagessen) und der geplanten Uhrzeit.	3	5	- Alle Geräte (nur 1 PF wird gleichzeitig dargestellt) - einfach, sequentiell, Darstellung gut skalierbar - keine Einschränkungen
	Ernährungsplan verwalten (komplex)	Auswählen von Einträgen, die in einer stundenplanähnlichen Struktur abgelegt sind.	2	6	- Nur für Windows Mobile - komplexe, effiziente Bedienung, erfordert feinmotorische Fähigkeiten - persönliche Daten (bei Personen in der Umgebung zu meiden)
	Rezept erstellen (einfach)	Definieren eines Rezeptes bestehend aus: Bild, Kurzbeschreibung, Zubereitung, Zutatenliste, Einordnung in die Lebensmittelkategorie	7	9	- Alle Geräte (2 PF müssen gleichzeitig darstellbar sein) - einfacher sequentieller Ablauf - keine Einschränkungen
	...				

Tabelle 10: Ausschnitt einer „Domain Theory“ am Beispiel des Adipositas-Begleiters

Beschrieben werden die einzelnen Benutzerschnittstellenfragmente in Tabelle 10 anhand von sechs Merkmalen:

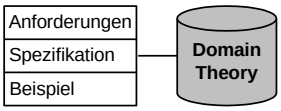
- Kategorie
  Ordnet jedes Benutzerschnittstellenfragment einer Kategorie zu, welche eine Aussage über die Art und Nutzung des Benutzerschnittstellenfragmentes ermöglicht.



- Name
  Möglichkeit zur Identifizierung eines Benutzerschnittstellenfragmentes, wobei sich die Benutzerschnittstellenfragmente und auch die gewählten Namen an dem Szenario orientieren, wie es in Abschnitt 5.6 zur Definition der Interaktionsbeschreibung für den Adipositas-Begleiters eingeführt wurde.
- Beschreibung
  Beschreibt, was das Benutzerschnittstellenfragment leistet und aus welchen Teilen es besteht.
- #PF
  Anzahl der Präsentationsfragmente im Benutzerschnittstellenfragment.
- #DF
  Anzahl der Dialogfragmente im Benutzerschnittstellenfragment.
- Kontextvektor
  Informale Beschreibung der Eigenschaften des Kontextvektors.

Die Kategorisierung in „Primitive BSF“, „Domänenunabhängige BSF“ und „Domänenspezifische BSF“ soll veranschaulichen, dass eine „Domain Theory“ eine notwendige Menge an Benutzerschnittstellenfragmenten enthalten muss. Im Einzelnen werden sogenannte „Primitive BSF“ benötigt, die das Eingeben von Text (realisiert durch ein Input-Feld), die Auswahl aus einer definierten Menge (realisiert durch Verweislisten) und schließlich die Darstellung von Inhalten (realisiert durch Text und Bild) ermöglichen. Diese Benutzerschnittstellenfragmente definieren eine atomare Menge aus Interaktionen, die zwar für alle Nutzungskontext eingesetzt werden können, dort aber auch nur eine sehr primitive Benutzerschnittstelle realisieren. Die Bereitstellung solcher Benutzerschnittstellenfragmente ist dennoch wichtig, da diese als Basisbausteine fungieren, mit denen Interaktionen beschrieben werden können, für die es noch keine passenden Benutzerschnittstellenfragmente in der „Domain Theory“ gibt. „Domänenunabhängige BSF“ beschreiben eine Menge von Benutzerschnittstellenfragmenten, die in unterschiedlichen Anwendungsdomänen verwendet werden und allgemeine Konstrukte von Benutzerschnittstellen beschreiben, die unabhängig von der Anwendungsdomäne der „Domain Theory“ sind. „Domänenspezifische BSF“ sind schließlich alle BSF, die sich direkt auf die Anwendungsdomäne beziehen und spezifische Lösungsbausteine beinhalten. Beispiel für eine Zuordnung von Benutzerschnittstellenfragmente zu den Kategorien finden sich in Tabelle 10.

Betrachtet man die informale Beschreibung der Kontextvektoren (in Tabelle 10) so wird offensichtlich, dass die dort angegebene Beschreibung nur einen ersten Anhaltspunkt für die eigentlichen Kontextinformationen liefert. Es wurde versucht darzustellen, dass unterschiedliche Benutzerschnittstellenfragmente ähnliche Interaktionen abbilden können, und Bewertungsfunktionen notwendig sind, die anhand des Kontextvektors ein für den Anwendungsfall „optimales“ Benutzerschnittstellenfragment ermitteln. Der erste „Spiegelstrich“ in der Beschreibung des Kontextvektors gibt an, welche Einschränkungen es im Bezug auf das Endgerät gibt, während der zweite Spiegelstrich angiebt, welche Anforderungen an den Benutzer gestellt werden.



Der dritte beschreibt schließlich, welche Anforderungen das Benutzerschnittstellenfragment an die Umgebung der Nutzung stellt. Weiterhin enthält die Tabelle Angaben von #PF und #DF, die einen Anhaltspunkt für die Komplexität des Benutzerschnittstellenfragmentes gibt. Ziel ist es darzustellen, dass Benutzerschnittstellenfragmente nicht einfach nur Masken oder Widgets sind, die wieder verwendet werden, sondern tatsächlich „Teile eines Interaktionsablaufes“.

Betrachtet man die Definition der „Domain Theory“ so ergibt sich der Eindruck, dass diese eine primitive, unsortierte Menge von Elementen beschreibt, die lediglich durch die Informationen des Kontextvektors spezifiziert werden. Darüber hinaus beinhaltet der Kontextvektor keine Informationen, welche die Interaktionen beschreibt, die durch das Benutzerschnittstellenfragment realisiert werden. Erst durch die Betrachtung des Objektmetaphern-Kataloges wird dieses Bild vervollständigt, und den Benutzerschnittstellenfragmenten sowohl eine inhaltliche Beschreibung als auch eine semantische Bedeutung zugeordnet. Dabei ist zu berücksichtigen, dass jedes Benutzerschnittstellenfragment der „Domain Theory“ im OM-Katalog verzeichnet ist (vgl. Abschnitt 5.4).

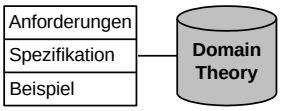
7.4 Zusammenfassung und abschließende Betrachtung

Für den in der Arbeit verwendeten Ansatz werden Bausteine (in diesem Kapitel Benutzerschnittstellenfragmente genannt) benötigt, aus denen Benutzerschnittstellen durch Komposition generiert werden können. Diese Bausteine müssen dabei primär zwei konkurrierende Anforderungen erfüllen: einerseits müssen sie detaillierte Informationen enthalten, um daraus eine ausführbare Benutzerschnittstelle realisieren zu können, andererseits müssen sie abstrakt genug sein, um in unterschiedlichen Benutzerschnittstellen als wieder verwendbarer Baustein eingesetzt zu werden.

Um diese Fragestellung zu konkretisieren wurden in einem ersten Schritt Anforderungen an diese Bausteine identifiziert. Hiezu wurde die allgemeine Fragestellung nach Bausteinen für Benutzerschnittstellen in die drei Teilbereiche Präsentation, Dialogsteuerung und schließlich Anwendungsschnittstelle unterteilt, und für jedes dieser drei Teile eingehend betrachtet.

Die Definition von Benutzerschnittstellenfragmenten sieht vor, dass diese als „abgrenzbare und zusammengehörige Ausschnitte aus einem Interaktionsablauf“ (vgl. Abschnitt 7.2.1) definiert werden. Diese Definition orientiert sich an dem Interaktionskonzept, welches in Abschnitt 5.1.3 beschrieben wurde, und stellt sicher, dass Benutzerschnittstellenfragmente auf die Interaktionsbeschreibung abgebildet werden können.

Die formale Spezifikation von Benutzerschnittstellenfragmenten orientiert sich an den drei Teilbereichen einer Benutzerschnittstelle (Präsentation, Dialogsteuerung und Anwendungsschnittstelle) und spezifiziert gleichsam eine „Event-basierte“ Umgebung zur Ausführung der Benutzerschnittstellenfragmente. Eine Realisierung dieser Umgebung kann auf Basis nahezu jeder Programmiersprache erfolgen, da nur einfache Konstrukte (Events & Event-Handler) vorausgesetzt werden.



Die Spezifikation von Benutzerschnittstellenfragmenten ermöglicht in einem weiteren Schritt die Definition der „Domain Theory“ als eine ungeordnete Menge von Benutzerschnittstellenfragmenten, die lediglich durch einen Kontextvektor beschrieben werden. Dabei ist zu beachten, dass der Kontextvektor keine beschreibenden Informationen über die Interaktionen enthält, die durch dieses Benutzerschnittstellenfragment realisiert werden. Semantische Beschreibungen werden ausschließlich über den OM-Katalog verwaltet. Dadurch wird eine Kategorisierung und Identifizierung der Benutzerschnittstellenfragmente ermöglicht.

Im Abschluss wurde ein Ausschnitt aus einer „Domain Theory“ für den Anwendungsfall des Adipositas-Begleiter vorgestellt und diskutiert, wie mögliche Benutzerschnittstellenfragmente aussehen können und welche von diesen notwendiger Bestandteil einer „Domain Theory“ sind.

8 Generierung der Benutzerschnittstelle am Beispiel des Adipositas-Begleiters

In Kapitel 4 wurden im Rahmen der Entwicklung eines Blueprints Modelle und Verfahren definiert, die zur Verfügung zu stellen sind, um eine automatisierte Benutzerschnittstellengenerierung nach dem Leitbild des „Compositional Modeling“ zu ermöglichen. Abbildung 38 greift das in diesem Zusammenhang vorgestellte Modell der Bausteine des Generierungsprozesses (vgl. Abbildung 13) auf und visualisiert (blau hervorgehoben) die Teile, die innerhalb dieses Kapitels erarbeitet werden.

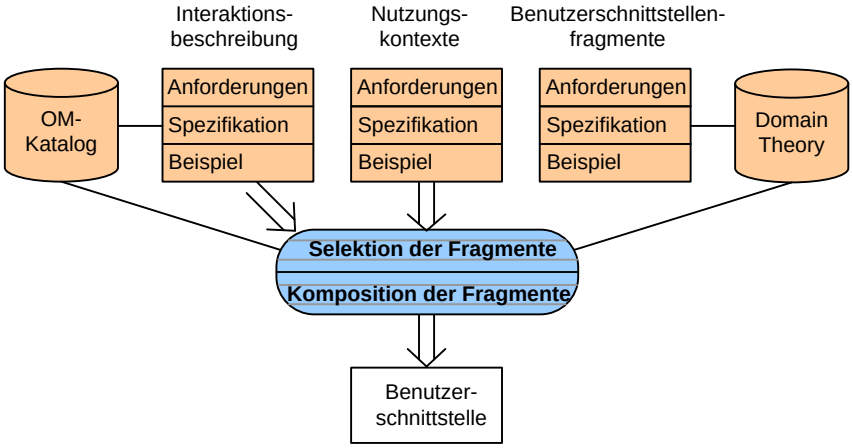


Abbildung 38: Einordnung der Generierung in den CM-Prozess

Ziel dieses Kapitels ist die Definition von Verfahren und Methoden zur Generierung von Benutzerschnittstellen basierend auf den im Vorfeld spezifizierten Modellen (Interaktionsbeschreibung, Nutzungskontexte und Benutzerschnittstellenfragmente) und Katalogen (OM-Katalog und Domain Theory).

In einem ersten Schritt wird der Kompositionsprozess, wie er in Kapitel 4 grob umrissen wurde, detailliert diskutiert und Teilaufgaben, Methoden, Rollen und ein Vorgehen basierend auf den entwickelten Modellen und Katalogen definiert. In Analogie zum „Compositional Modeling“ wird dabei ein zweistufiges Verfahren entwickelt, das in einem ersten Schritt die Identifikation von Benutzerschnittstellenfragmenten aus der Domain Theory vorsieht, mit denen die Interaktionsbeschreibung unter den Vorgaben des Nutzungskontextes umgesetzt werden soll. Im zweiten Schritt des Kompositionsprozesses werden die ermittelten Benutzerschnittstellenfragmente unter Berücksichtigung der Anforderungen komponiert, die in der Interaktionsbeschreibung modelliert wurden. Beide Schritte werden beschrieben und Methoden und Algorithmen konzipiert, welche für die Durchführung notwendig sind. Am Beispiel des Adipositas-Begleiters wird die konkrete Arbeitsweise des Verfahrens erläutert und exemplarisch ein vollständiger Generierungsprozess durchgeführt, der zum Abschluss des Kapitels bewertet wird.

8.1 Konzeption des Kompositionsprozesses

Ausgangslage für die Generierung der Benutzerschnittstelle ist die Aufgabenstellung nach Falkenhainer und Forbus, nach der basierend auf der „scenario description", der „domain theory" und der „query" das Zielmodell („scenario model") generiert wird (vgl. [FF91]). In den vorhergehenden Kapiteln (Kapitel 5 - 7) dieser Arbeit wurden in Analogie zum „Compositional Modeling" Modelle und Kataloge für die Benutzerschnittstellengenerierung erarbeitet. Basierend auf diesen Modellen lässt sich die Aufgabenstellung wie folgt formulieren:

> *Gegeben seien eine Interaktionsbeschreibung in Form eines Abstract Interaction Nets (definiert in Abschnitt 5.3), eine „Domain Theory" (definiert in Abschnitt 7.3) bestehend aus wieder verwendbaren Benutzerschnittstellenfragmenten (definiert in Abschnitt 7.2.1) und ein Nutzungsprofil (definiert in Abschnitt 6.3).*
>
> *Aufgabenstellung ist es, basierend auf den Vorgaben der Interaktionsbeschreibung durch Komposition der Benutzerschnittstellenfragmente der „Domain Theory" eine ausführbare Benutzerschnittstelle (definiert in Abschnitt 7.2.2.2) zu generieren, welche den durch das Nutzungsprofil definierten Anforderungen bestmöglich genügt.*

Die Interaktionsbeschreibung liefert zu dieser Aufgabenstellung Informationen über die Benutzerschnittstelle, indem sie Interaktionen und deren Abläufe basierend auf Interaktionsmetaphern spezifiziert. Die „Domain Theory" stellt in diesem Rahmen eine reine Wissensbasis bereit, die zwar domänenabhängig ist, aber keine Informationen über die zu generierende Benutzerschnittstelle liefert. Das Nutzungsprofil definiert schließlich eine Bewertungsgrundlage für die Generierung und legt fest, wie Interaktionen realisiert werden müssen und auch welche Benutzerschnittstellenfragmente dafür geeignet sind. Betrachtet man die gegebene Aufgabenstellung eingehender, so lassen sich zwei Teilaufgaben für den Kompositionsprozess identifizieren:

- Abbildung der Interaktionsmetaphern auf Benutzerschnittstellenfragmente
  Die Interaktionsbeschreibung liegt in Form eines AINs vor und basiert somit auf Objekt- und Interaktionsmetaphern, die miteinander in Beziehung stehen. In einem ersten Schritt gilt es, diese Elemente (Objekt-, Interaktionsmetaphern und ihre Beziehungen zueinander) auf die verfügbare Wissensbasis (die Benutzerschnittstellenfragmente in der „Domain Theory") abzubilden. Sowohl die Interaktionsbeschreibung als auch die Benutzerschnittstellenfragmente beziehen sich auf Interaktionsabläufe. Folglich lässt sich auch jedes Benutzerschnittstellenfragment durch ein AIN beschreiben. Für eine Abbildung müssen in der Interaktionsbeschreibung Teilbereiche (Ausschnitte des AINs) identifiziert werden, die mit den in der Domain Theory verfügbaren Benutzerschnittstellenfragmenten abbildbar sind. Die Auswahl der Teilbereiche und damit einhergehend auch die Menge der in Frage kommenden Benutzerschnittstellenfragmente zur Abbildung der Interaktionsbeschreibung müssen mit Hilfe des Nutzungsprofils bewertet werden. Angestrebt wird eine optimierte Auswahl an Benutzerschnittstellenfragmenten, die sich dadurch auszeichnet, dass die Menge aller Benutzerschnittstellenfragmen-

te, die zur Abbildung der Interaktionsbeschreibung ausgewählt wurden, den Anforderungen im Nutzungsprofil am besten genügen.

Diese ganzheitliche (auf die gesamte Interaktionsbeschreibung ausgerichtete) Formulierung der Aufgabenstellung ist notwendig, da eine „lokale Optimierung“ einzelner Benutzerschnittstellenfragmente nicht notwendigerweise für die gesamte Interaktionsbeschreibung die beste Lösung darstellt.

- Komposition der ausgewählten Benutzerschnittstellenfragmente

  Wurde eine passende Auswahl an Benutzerschnittstellenfragmenten festgelegt, welche die Interaktionsbeschreibung vollständig abbilden, so muss diese in einem zweiten Schritt zu einer ausführbaren Benutzerschnittstelle komponiert werden. Als Informationsquelle steht hierfür die Interaktionsbeschreibung zur Verfügung, welche die Beziehungen zwischen Interaktionen (beispielsweise Interaktionsfolgen, Verzweigungen oder Parallelität) beschreibt, die auch bezogen auf die Benutzerschnittstellenfragmente immer noch Gültigkeit haben. Neben diesen Anforderungen muss die Komposition auch die Eigenschaften des Endgerätes zur Präsentation der Benutzerschnittstellenfragmente und schließlich auch die Fähigkeit des Benutzers zur Bedienung der Benutzerschnittstelle berücksichtigen. Die Fragestellung, welche der Benutzerschnittstellenfragmente gleichzeitig verfügbar sind, ist abhängig von diesen Anforderungen. Zusätzlich sind aber auch Datenflüsse zwischen den Benutzerschnittstellenfragmenten zu berücksichtigen. Insbesondere die „Layoutpattern“ Komponente des Nutzungsprofils (vgl. Abschnitt 6.3.5) liefert Informationen über die verfügbaren Bereiche auf einer graphischen Benutzerschnittstelle, und legt damit auch fest, wie viele Benutzerschnittstellenfragmente gleichzeitig präsentiert werden können. Basierend auf diesen Anforderungen lässt sich die Aufgabenstellung der Komposition wie folgt fokussieren: komponiere die Menge der Benutzerschnittstellenfragmente unter Berücksichtigung der Vorgaben der Interaktionsbeschreibung, der Anforderungen des Nutzungsprofils (insbesondere des verwendeten „Layoutpatterns“) und der Datenflüsse zwischen den Benutzerschnittstellenfragmenten.

In Abbildung 39 wird der Generierungsprozess detaillierter visualisiert und in den Kontext des vorhergehenden (die Modellierung) und des nachfolgenden Arbeitschrittes (Evaluation und Konfiguration) gestellt. Eingeteilt ist die Abbildung in drei horizontale Ebenen. Die oberste beschreibt die groben Arbeitsschritte, den Ablauf und die beteiligten Rollen. Die zweite Ebene verfeinert die Aktivitäten und stellt Beziehungen zu Modellen und Wissensbasen her. Die Wissensbasen definieren schließlich den „Domänenspezifischen Baukasten für die Benutzerschnittstellengenerierung“ und ist auf der untersten Ebene abgebildet.

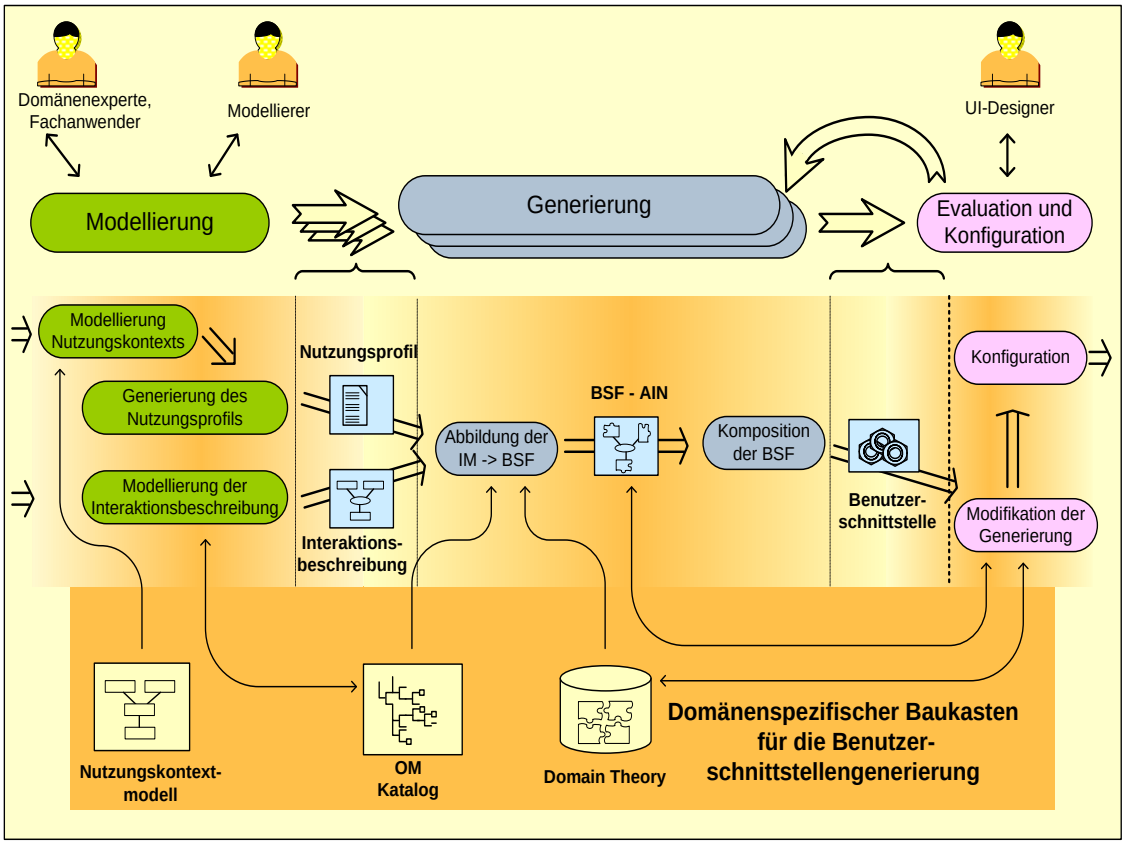


Abbildung 39: Entwicklungsprozess zum Modellierungs- und Generierungsprozess

Wie in der ersten Ebene in Abbildung 39 dargestellt, sind drei Rollen für den Generierungsprozess identifiziert worden. Fachanwender und Domänenexperten definieren eine Gruppe von Personen mit höchstem Know-how im Domänenbereich, bei denen aber keine speziellen Anforderungen in Bezug auf ihr IT-technisches Verständnis gestellt werden. Primäre Aufgabe dieser Gruppe ist der Aufbau des „Domänenspezifischen Baukastens“ aus der fachlichen Perspektive und die Modellierung der Interaktionsbeschreibung. In der Domäne der Telemedizin wird diese Rolle durch die Personengruppe der Fachärzte, Klinik-Personal aber auch pflegerisch tätiger Berufsgruppen gestellt.

Unterstützt wird die Gruppe der Domänenexperten und Fachanwender in ihren Aufgaben durch die Rolle des Modellierers. Ausgefüllt wird diese Rolle durch Software-Architekten bzw. IT-Experten aus dem Bereich des „Requirement Engineerings“, deren Aufgabe die Formalisierung des Experteninputs in Modellen und Repositories ist.

Der User Interface Designers (UI-Designer in Abbildung 39) ist schließlich für die Analyse und Konfigurationen des Generierungsergebnisses verantwortlich. Sie modifiziert Benutzerschnittstellenfragmente und stößt gegebenenfalls die Generierung erneut an.

Die sich überlappenden Pfeile auf der ersten Ebene, die von der Modellierungs- auf die Generierungsphase zeigen, sollen veranschaulichen, dass nach einmaliger Modellierung der Interaktionsbeschreibung die

Generierung mehrfach, nämlich jeweils für ein Nutzungsprofil angestoßen wird. Der Arbeitschritt „Generierung“ und auch die „Evaluation und Konfiguration“ beziehen sich dabei immer auf das Ergebnis genau einer Generierung. Zwischen „Generierung“ und „Evaluation und Konfiguration“ wurde weiterhin ein Kreislauf modelliert, der darstellen soll, dass auf Basis der Erkenntnisse der Evaluation mit einer modifizierten Ausgangsbasis wiederholt generiert werden kann, bis das gewünschte Ergebnis erreicht wurde. Gesteuert wird dieser Kreislauf durch die Rolle des „UI-Designers“.

Die zweite Ebene in Abbildung 39 detailliert die einzelnen Arbeitsschritte. So erfolgt im Arbeitsschritt „Modellierung“ die Spezifikation von Nutzungskontexten und daraus schließlich die Entwicklung von Nutzungsprofilen und die Modellierung der Interaktionsbeschreibung.

Im Rahmen der „Generierung“ sind die beiden Teilschritte „Abbildung der Interaktionsmetaphern auf Benutzerschnittstellenfragmente“ und „Komposition der Benutzerschnittstellenfragmente“ durchzuführen. Die Informationsbasis für beide Schritte liefert die Modellierungsphase, nämlich das Nutzungsprofil und die Interaktionsbeschreibung. Der erste Teilschritt „Abbildung IM->BSF“ (in Abbildung 39) bezieht weiterhin den OM-Katalog und die „Domain Theory“ als zusätzliche Informationsquellen mit ein, während der zweite Generierungsschritt „Komposition der BSF“ (in Abbildung 39) als Input das Ergebnis des ersten Schrittes erhält. In Abbildung 39 ist dieser Input als „BSF-AIN“ gekennzeichnet, und soll verdeutlichen, dass es sich dabei um die Interaktionsbeschreibung (in Form eines AIN) handelt, auf die Benutzerschnittstellenfragmente (BSF) der „Domain Theory“ abgebildet wurden. Eine Definition des BSF-AIN erfolgt in Abschnitt 8.2.1.

Im Rahmen des Arbeitsschrittes „Evaluation und Konfiguration“ wurden auf der zweiten Ebene die „Modifikation der Generierung“ und die „Konfiguration“ als Teilarbeitsschritte identifiziert. Während die Konfiguration das „Feintuning“ der Benutzerschnittstelle beschreibt, beinhaltet der Arbeitsschritt „Modifikation der Generierung“ den auf der ersten Ebene beschriebenen Kreislauf der wiederholten Generierung, wenn die Benutzerschnittstelle nicht den Anforderungen des UI-Designers entspricht. Hierzu kann der UI-Designer manuell in den Generierungsprozess eingreifen, indem er die Domain Theory um neue Benutzerschnittstellenfragmente erweitert, und/oder das BSF-AIN manuell modifiziert, um in den Interaktionsablauf einzugreifen, oder andere Benutzerschnittstellenfragmente für die Realisierung der Interaktionsmetaphern heranzuziehen. Aus der Perspektive des gesamten Generierungsprozesses muss der Umgang mit manuellen Eingriffen in einem weitgehend automatisierten Prozess kritisch hinterfragt werden. Handelt es sich bei diesen Eingriffen um Modifikationen, die sich ausschließlich auf den „domänenspezifischen Baukasten“ auswirken, so kann gewährleistet werden, dass bei einer wiederholten Generierung auch das gewünschte Ergebnis erreicht wird. Modifikationen, die am BSF-AIN durchgeführt werden, können dabei nicht mehr reproduziert werden, so dass diese Veränderungen im Zuge einer erneuten Generierung verloren gehen. Um dieser Problemstellung entgegen zu wirken, wird die Benutzerschnittstelle, die auf Basis des modifizierten BSF-AIN generiert wurde, als Baustein in die Domain Theory aufge-

nommen und mit einem Nutzungskontextvektor[37] versehen, der ein ideales Matching zum gegebenen Nutzungsprofil aufweist. Weiterhin wird der OM-Katalog um das AIN erweitert, welches die Interaktionsbeschreibung modelliert. Im Falle der erneuten Generierung wird das modifizierte BSF-AIN (vgl. „Algorithmus zum Aufbau des BSF-AINs“ in Abschnitt 8.2.2) als geeignetes Benutzerschnittstellenfragment identifiziert, so dass die getätigten Modifikationen erhalten bleiben.

Der „domänenspezifische Baukasten für die Benutzerschnittstellengenerierung“ (dargestellt auf der untersten Ebene in Abbildung 39) definiert die Wissensbasis für die Generierung von Benutzerschnittstellen einer Anwendungsdomäne.

In den folgenden Abschnitten werden die beiden Teilschritte im Rahmen der Generierung „Abbildung der Interaktionsmetaphern auf Benutzerschnittstellenfragmente“ und „Komposition der Benutzerschnittstellenfragmente“ eingehend betrachtet. Für jeden der Teilschritte werden Werkzeuge und Verfahren definiert, die eine automatisierte Durchführung ermöglichen. Schließlich wird der Generierungsprozess an einem Ausschnitt aus dem Anwendungsbeispiel des Adipositas-Begleiters exemplarisch durchgeführt und im Anschluss diskutiert.

[37] Nutzungskontextvektoren beschreiben Eigenschaften eines BSF in der „Domain Theory“ auf Basis der Struktur eines Nutzungsprofils (vgl. Abschnitt 7.3).

8.2 Generierung des BSF-AIN

Die Generierung der Benutzerschnittstelle folgt einem zweischrittigen Verfahren (vgl. Abschnitt 8.1). Die folgenden Abschnitte beschreiben den ersten Generierungsschritt, in welchem die verfügbare Wissensbasis (nämlich die Elemente der „Domain Theory“) auf die Interaktionsbeschreibung abgebildet wird. Diese Abbildung wird durch ein sogenanntes „BSF-AIN“ beschrieben, das durch eine Datenstruktur realisiert wird, die auf den Formalismen des AIN beruht. Hierzu werden in einem ersten Schritt die Anforderungen an ein BSF-AIN beschrieben und im Anschluss ein Algorithmus präsentiert, der in der Lage ist, ein BSF-AIN automatisiert zu errechnen. Eine Kernfunktion dieses Algorithmus ist eine Bewertungsfunktion, die es erlaubt, eine Menge von Benutzerschnittstellenfragmenten mit einer anderen Menge auf Basis des Nutzungsprofils zu vergleichen, und zu berechnen, welche der beiden Mengen eine geeignetere Realisierung der Interaktionsbeschreibung ermöglicht.

8.2.1 Zielsetzung bei der Generierung des BSF-AIN

Die Interaktionsbeschreibung wird durch ein AIN spezifiziert, das Interaktionsabläufe mit Hilfe von Objektmetaphern und Interaktionen mit diesen Metaphern modelliert. Dem gegenüber steht die „Domain Theory“, die durch eine Sammlung von Benutzerschnittstellenfragmenten definiert ist. Jedes Benutzerschnittstellenfragment realisiert einen Teil eines Interaktionsablaufes und liefert eine implementierungsnahe Beschreibung. Gesucht ist eine Lösung, die folgende Anforderungen erfüllt:

- Es soll eine Menge von Benutzerschnittstellenfragmenten ermittelt werden, wobei jedes Benutzerschnittstellenfragment einen Teil der Interaktionsbeschreibung abbildet.
- Die Interaktionsbeschreibung muss vollständig und überschneidungsfrei durch diese Menge abgebildet werden.
- Die Menge der Benutzerschnittstellenfragmente muss im Bezug auf das Nutzungsprofil „optimal“ sein. Optimal ist die Menge dann, wenn keine andere Menge gebildet werden kann, welche den durch das Nutzungsprofil definierten Anforderungen besser gerecht werden kann.
- Zusätzlich zur gesuchten Menge der Benutzerschnittstellenfragmente muss eine Abbildung angegeben werden, welche jedes Element der Menge auf den Teil des AINs abbildet, das durch dieses Element realisiert wird. Im Folgenden wird der Begriff BSF-AIN für die Beschreibung dieser Lösung verwendet.

Eine universelle Lösungsstrategie für die gegebene Problemstellung ist als komplex einzustufen. Einerseits muss die Interaktionsbeschreibung in Teile (überlappungsfreie Ausschnitte des AINs) zerlegt werden. Jedes dieser Teile kann potentiell mit einem verfügbaren Benutzerschnittstellenfragment abgebildet werden. Die Auswahl der Teile einerseits und die Auswahl der Benutzerschnittstellenfragmente andererseits muss weiterhin mit Hilfe des Nutzungskontextvektors des Benutzerschnittstellenfragmentes und dem Nutzungsprofil für die Interaktionsbeschreibung bewertet werden.

Eine Voraussetzung ist die Verfügbarkeit von Werkzeugen, mit deren Hilfe die Frage beantwortet werden kann, was für ein AIN durch ein Benutzerschnittstellenfragment in der Domain Theory realisiert wird. Erst wenn so ein Werkzeug verfügbar ist, kann beispielsweise mit Hilfe von Graph-Matching Algorithmen untersucht werden, ob dieses AIN auch in der Interaktionsbeschreibung enthalten ist. Zusätzlich bleibt die eigentliche Problemstellung (vollständige Abdeckung der Interaktionsbeschreibung und optimale Auswahl der Menge der Benutzerschnittstellenfragmente) bestehen.

In dieser Arbeit wurde ein pragmatischer Ansatz zur Ermittlung des BSF-AINs gewählt, der durch den Einsatz des Objektmetaphern-Kataloges und der Forderung ermöglicht wird, dass jede Objekt- und Interaktionsmetapher der Interaktionsbeschreibung auch im Objektmetaphern-Katalog detailliert ist. Der folgende Abschnitt beschreibt ein Verfahren, das auf diesem Ansatz beruht.

8.2.2 Algorithmus zum Aufbau des BSF-AIN

Der in dieser Arbeit verwendete Algorithmus zum Aufbau des BSF-AINs nutzt die hierarchische Strukturierung der Interaktionsmetaphern im Objektmetaphern-Katalog zur Identifikation möglicher Benutzerschnittstellenfragmente in der Interaktionsbeschreibung. Bei der Erstellung der Interaktionsbeschreibung ist der Modellierer angehalten, möglichst Elemente aus dem Objektmetaphern-Katalog wieder zu verwenden, oder (falls neue Elemente benötigt werden) den Katalog zu erweitern. Dabei wird vorausgesetzt, dass möglichst komplexe Elemente ausgewählt werden, um zu verhindern, dass Interaktionen redundant modelliert werden (vgl. Abschnitt 5.4). Durch die Definition von „Abstrakten Interaktionen“, „Alternativen“ und „Varianten“ im Objektmetaphern-Katalog wurde weiterhin die Möglichkeit geschaffen, auch „Platzhalter“ zu modellieren, bei denen der Modellierer nicht genau spezifizieren will oder kann, wie eine Realisierung aussehen muss. Der Einsatz von Objektmetaphern unterstützt dieses Vorgehen, da leichter zu identifizieren ist, welche Semantik mit der Interaktion verbunden wird.

Der hierarchische Aufbau zieht sich als Konzept durch den gesamten Objektmetaphern-Katalog. So wird jede Interaktionsmetapher (sei es nun eine „Abstrakte Interaktion“, eine „Alternative“ oder „Variante“) schließlich durch ein AIN beschrieben. Die Möglichkeit, bei der Modellierung bewusst auf Abstrakte Interaktionen (Modellierer will mehr als eine Alternative anbieten), Alternativen (auf eine Interaktionsmethode reduziert, wobei es vom Nutzungsprofil abhängt, wie diese umgesetzt wird) oder Varianten (genau der im AIN beschriebene Lösungsweg) zuzugreifen, führt eine weitere hierarchische Strukturierung bei der Beschreibung von Interaktionsmetaphern ein. Erst Interaktionsmetaphern im Objektmetaphern-Katalog, zu denen kein AIN angegeben wurde, können als elementare Bausteine aufgefasst werden.

Basierend auf diesen Überlegungen liegt es also nahe, festzustellen, dass im Objektmetaphern-Katalog keine Interaktionsmetapher existiert, die einen Ausschnitt der Interaktionsbeschreibung umfasst, der mehr als eine Interaktionsmetapher beinhaltet. Folglich muss nicht überprüft werden, ob Teile der Interaktionsbeschreibung durch Interaktionsmetaphern des Objektmetaphern-Kataloges realisiert werden. Dies bedeutet, dass es auch kein Benutzerschnittstellenfragment geben kann, das mehr als eine Interaktionsmetapher der Interaktionsbeschreibung realisiert.

Sollte es Benutzerschnittstellenfragmente geben, die mehrere Interaktionsmetaphern in der Interaktionsbeschreibung abdecken, so muss auch eine Interaktion im Objektmetaphern-Katalog existieren, welche genau diese beschreibt. Gewährleistet wird dies durch die Definitionen der Beziehungen zwischen dem OM-Katalog und der Domain-Theory (vgl. Abschnitt 5.4 und 7.3).

Eine Definition der Beziehungen zwischen OM-Katalog und „Domain Theory“, die eine stärkere Entkopplung der beiden Wissensbasen ermöglicht, würde zu einer höheren Flexibilität der Wissensbasen und zur Verbesserung der Erweiterbarkeit des Generierungsprozesses führen. Konzepte und Lösungsansätze für solch eine Definition und auch die notwendigen Werkzeuge und Methoden, welche ein automatisiertes Matching ermöglichen würden, übersteigen bei weitem den Umfang dieser Arbeit, sind aber als Fortsetzung und Weiterführung durchaus wünschenswert.

Legt man diese Restriktion zu Grunde, so kann ein Algorithmus angegeben werden, der in einem ersten Schritt mit einer isolierten Analyse der Interaktionsmetaphern der Interaktionsbeschreibung beginnt.

Im Folgenden ist die zentrale Methode „()bildeBSF-AIN()“ in „Pseudocode[38]“ skizziert, die aus einer gegebenen Interaktionsbeschreibung, dem Nutzungsprofil und dem OM-Katalog ein BSF-AIN erzeugt.

```
(BSF-AIN[], AIN) bildeBSF-AIN (OM-KATALOG omKatalog, PROFIL nutzungsprofil, AIN ain)

IM[] interaktionen = ain.AlleInteraktionen() // liefert alle Interaktionsmetaphern des AINs
BSF-AIN[] abbildung = {} // Eine Liste von Tupeln (Interaktionsmetapher, BSF)
BSF/AIN ergebnis = {} // Element aus dem OM-Katalog und somit entweder ein AIN oder ein Verweis auf ein BSF
WHILE (interaktionen.count() != 0 ) DO //Schleife solange durchlaufen bis die Liste leer ist.
        ergebnis = analysiereInteraktion(omKatalog, nutzungsprofil, interaktionen[1]) // Das Ergebnis dieser Methode ist entweder ein BSF oder ein AIN, das die Interaktionsmetapher unter Berücksichtigung des Nutzungsprofils realisiert.
        IF ( ergebnis.istBSF() ) // Wenn „ergebnis“ vom Typ BSF ist liefert die Bedingung ein „wahr“
                THEN
                        abbildung = abbildung + (interaktionen[1], ergebnis) //Erweitere die Liste um ein neues Tupel
                ELSE    //Das „ergebnis“ ist ein AIN
                        ain = erweitereAIN(ain, interaktionen[1], ergebnis)
                                //Ersetzte die Interaktion „interaktion[n]“ durch das AIN im „ergebnis“
                        interaktionen = interaktionen.addInteraktionen(ergebnis.AlleInteraktionen())
                                //Liste um alle Interaktionsmetaphern des AIN ergebnis erweitern
        interaktionen = interaktionen.entferne(interaktionen[1])
                        //Entfernt die übergebene Interaktion aus der Liste
RETURN (abbildung, ain)
```

[38] Der hier verwendete Pseudocode ist eine Mischung aus natürlicher Sprache und einer an Java angelehnten Programmiersprache.

Als Übergabeparameter erhält diese Methode den Objektmetaphern-Katalog, ein Nutzungsprofil und das zu analysierende AIN (die Interaktionsbeschreibung). Das Ergebnis dieser Funktion ist das gesuchte BSF-AIN. Es genügt den Anforderungen aus Abschnitt 8.2.1 und definiert somit das Ergebnis des ersten Kompositionsschrittes. Beschrieben wird das BSF-AIN durch die Datenstruktur BSF-AIN[] und ein AIN. Die Datenstruktur BSF-AIN [] enthält die Abbildung von Interaktionsmetaphern auf Benutzerschnittstellenfragmente in der Form$\{(IM_1,BSF_1),..,(IM_n,BSF_n)\}$. Weiterhin wird das BSF-AIN durch ein AIN beschrieben, das als eine erweiterte Variante der Interaktionsbeschreibung aufgefasst werden kann. Dieses AIN zeichnet sich dadurch aus, dass zu jeder Interaktionsmetapher des AINs ein BSF in der Datenstruktur BSF-AIN[] angegeben ist, das diese Interaktionsmetapher realisiert. Diese Beschreibung des BSF-AINs stellt einerseits sicher, dass die gesamte Interaktionsbeschreibung durch BSF abgebildet wurde, andererseits ist sicher gestellt, dass die Informationen über die Beziehungen der Interaktionsmetaphern aus der Interaktionsbeschreibung in Form des AIN weiterhin erhalten bleiben können. Die Komposition der BSF (zweiter Kompositionsschritt) muss sicherstellen, dass diese Beziehungen eingehalten werden.

Der Algorithmus sieht in einem ersten Schritt vor, dass eine Liste aller Interaktionsmetaphern des übergebenen AINs (Interaktionsbeschreibung) gebildet wird (ain.AlleInteraktionen()). Jede Interaktion dieser Liste wird nun separat über die Funktion „analysiereInteraktion()“ untersucht. Das Ergebnis dieser Funktion ist dabei entweder ein Benutzerschnittstellenfragment, das die Interaktion realisiert, oder aber ein AIN, das die Interaktion genauer spezifiziert. Anzumerken ist, dass es sich bei dem Ergebnis immer um das am besten geeignete BSF oder AIN bezogen auf das übergebene Nutzungsprofil handelt. „Am besten geeignet“ heisst dabei, dass für das BSF bzw. AIN die Bewertungsfunktion (vgl. Abschnitt 8.2.3) für das gegebene Nutzungsprofil den höchsten Wert ermittelt hat. Ein hoher Wert bedeutet, dass das BSF bzw. AIN den Anforderungen des Nutzungsprofils gerecht wird.

Wenn es sich bei dem Ergebnis um ein Benutzerschnittstellenfragment handelt, so wird die Ergebnisliste (BSF-AIN[]) um das Tupel (Interaktionsmetapher, Benutzerschnittstellenfragment) erweitert. Handelt es sich bei dem Ergebnis um ein AIN, so wird die untersuchte Interaktionsmetapher durch dieses ersetzt. Hierzu wird einerseits das übergebene AIN (die ursprüngliche Interaktionsbeschreibung) durch die Funktion „erweitereAIN()“ erweitert, indem die Interaktionsmetapher durch das AIN ersetzt wird. Weiterhin wird die Liste der zu untersuchenden Interaktionsmetaphern, um alle Interaktionsmetaphern dieses AINs erweitert. Die analysierte Interaktionsmetapher wird anschließend aus der Liste entfernt, da sie entweder in der Ergebnisliste BSF-AIN[] auf ein BSF abgebildet wurde, oder aber durch ein AIN ersetzt wurde, dessen Interaktionsmetaphern in die Liste der noch zu untersuchenden Interaktionsmetaphern aufgenommen wurden.

Der Algorithmus stoppt erst, nachdem die Liste aller zu untersuchenden Interaktionsmetaphern leer ist und damit alle Interaktionsmetaphern auf jeweils ein BSF abgebildet wurden.

Abbildung 40 visualisiert die Struktur der Aufrufe der einzelnen Funktionen zur Realisierung des Algorithmus und detailliert ihre Funktionsweise, indem es sie in Subfunktionen aufteilt. Die Rechtecke repräsentieren die Funktionen mit ihren Schnittstellen und die gerichteten Kanten die Aufrufe innerhalb der Funktionen. Wie aus dem obigen Pseudocode ersichtlich, wird von der Funktion „bildeBSF-AIN()“ sowohl die Funktion „erweitereAIN()“, als auch „analysiereInteraktion()“ aufgerufen. Im Folgenden wird anhand dieser Aufrufstruktur der Algorithmus beschrieben. Eine vollständige Spezifikation inklusive der Definition der verwendeten Datenstrukturen findet sich in Anhang C.

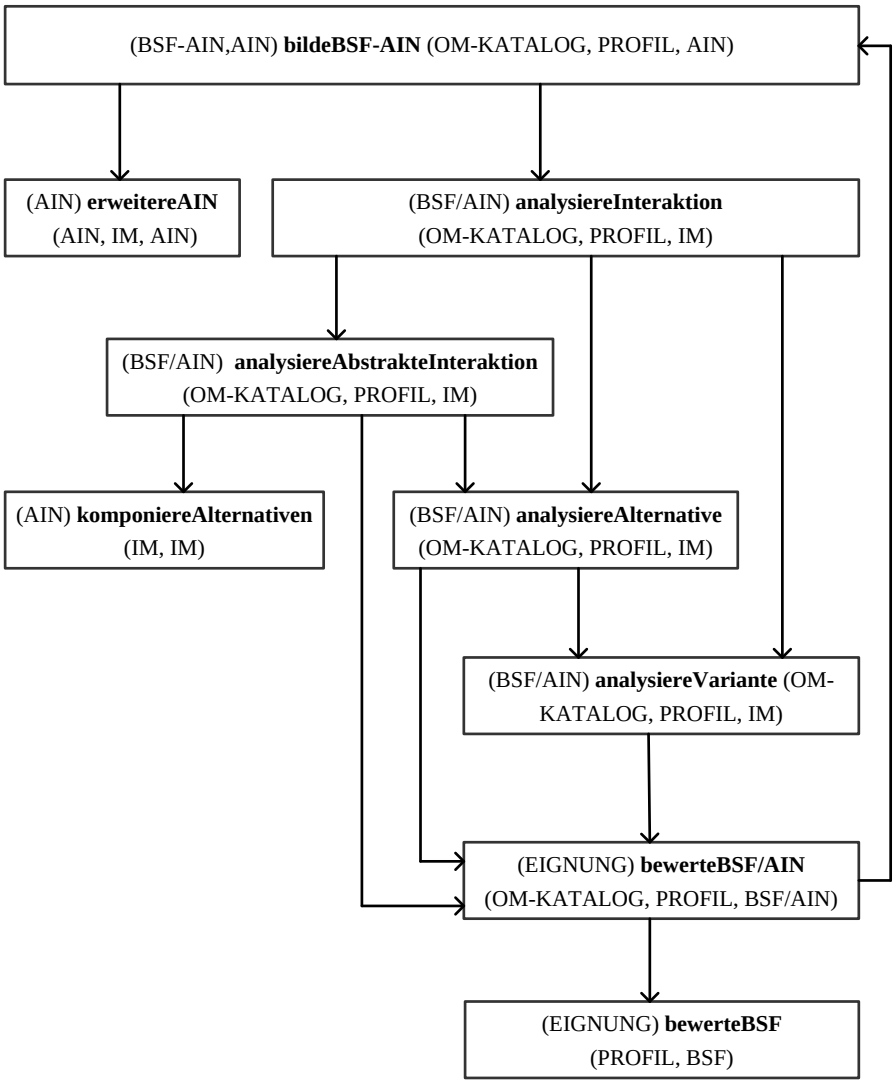


Abbildung 40: Aufrufstruktur der Funktion „bildeBSF-AIN()“

Die Funktion **„erweitereAIN(AIN, IM, AIN)“** erhält als Übergabeparameter ein AIN (zur besseren Identifikation im Folgenden startAIN genannt), eine Interaktionsmetapher und ein weiteres AIN (zur besseren Identifikation imAIN genannt), das diese Interaktionsmetapher spezifiziert. Das Ergebnis der Funktion ist ebenfalls ein AIN (zur besseren Identifikation zielAIN genannt). Aufgabe dieser Funktion ist es, im startAIN die übergebene Interaktionsmetapher durch das imAIN zu ersetzen und das neu entstandene zielAIN

als Ergebnis zurück zu geben. Die Realisierung solch einer Funktion ist basierend auf der vorliegenden Definition eines AINs trivial. Die Interaktionsmetapher kann als eine komplexe Transition in dem start-AIN aufgefasst werden, die durch das imAIN genauer spezifiziert wird. Komplexe Transitionen beschreiben eine Kerneigenschaft von AINs und wurden in Abschnitt 5.3.2 eingehend beschrieben.

Aufgabe der Funktion **„analysiereInteraktion()“** ist es, für eine übergebene Interaktionsmetapher („IM“ in Abbildung 40) eine bezüglich des Nutzungsprofils („PROFIL“ in Abbildung 40) optimale Abbildung zu finden. Als Ergebnis liefert die Funktion dabei entweder ein Benutzerschnittstellenfragment oder ein AIN zurück. Zu interpretieren ist das Ergebnis wie folgt: Wenn ein BSF zurückgegeben wurde, so stellt dieses die optimale Realisierung der Interaktionsmetapher dar. Im Falle eines AINs weiß man, dass eine bessere Realisierung der Interaktionsmetapher in dem zurückgegebenen AIN zu finden ist.

Realisiert wird diese Funktion indem in einem ersten Schritt überprüft wird, was für einen Typ die Interaktionsmetapher besitzt. Mit Hilfe des OM-Kataloges wird ermittelt, ob es sich um eine Abstrakte Interaktion, eine Alternative oder Variante handelt. Je nachdem, welcher Typ vorliegt, wird die Aufgabenstellung an die entsprechende Analysefunktion (analysiereAbstrakteInteraktion(), analysiereAlternative() oder analysiereVariante()) weitergereicht, in welcher die weitere Verarbeitung stattfindet.

Im Falle der Funktion **„analysiereAbstrakteInteraktion()“** wird im OM-Katalog überprüft, ob mehr als eine Alternative vorhanden ist. Ist dies der Fall, so wird über die Funktion „analysiereAlternative()“ für jede der Alternativen eine optimale Realisierung (AIN oder BSF) ermittelt und mit Hilfe der Funktion „bewerteAIN/BSF()“ bewertet. Die Funktion liefert einen numerischen Wert, der die einzelnen Alternativen miteinander vergleichbar macht.

Nun wird anhand des Nutzungsprofils überprüft, ob mehr als eine Alternative im Interaktionsablauf angeboten werden soll. In dem für den Anwendungsfall „Adipositas-Begleiter“ entwickelten Algorithmus werden drei Fälle unterschieden:

1. Nutzer möchte einen einfachen, sequentiellen Ablauf.
   Es wird keine Alternative (also nur die beste Lösung) angeboten.
2. Nutzer möchte möglichst flexibel und effizient interagieren.
   Es werden alle Alternativen angeboten, deren Eignung einen gewissen Schwellenwert überschreitet.
3. Keine klaren Präferenzen.
   Es werden zwei Alternativen angeboten, aber nur dann, wenn die zweitbeste Lösung einen gewissen Schwellenwert überschreitet.

Zur Unterscheidung der drei Fälle werden die Parameter „Interactionflow. sequential.Preference“, „Interactionflow. parallel.Preference“, „Interactionflow.mutable.Preference“ und „Interactionflow.efficient.Preference“ des Nutzungsprofils herangezogen (vgl. Funktion in Anhang C). Eine genauere Betrachtung dieser Parameter findet im Rahmen der Betrachtung der Bewertungsfunktion von Benutzerschnittstellenfragmenten in Abschnitt 8.2.3 statt.

Die Begrenzung auf die drei vorgestellten Fälle, aber auch die Abgrenzung voneinander, basiert auf ersten Erfahrungswerten bei der Generierung von Benutzerschnittstellen im Rahmen des Adipositas-Begleiters. Komplexere Varianten, die eingehend untersucht werden müssen, sind zukünftig wünschenswert und können die Flexibilität der Generierung weiter verbessern.

Soll mehr als eine Alternative angeboten werden, so werden mit Hilfe der Funktion **„komponiereAlternativen()“** die Alternativen zu einem AIN komponiert. Die Realisierung der Komposition kann als trivial angesehen werden, da alle Alternativen derselben Abstrakten Interaktion zugeordnet sind und folglich über dieselben Schnittstellen verfügen. Abbildung 41 veranschaulicht anhand eines Beispiels die Komposition.

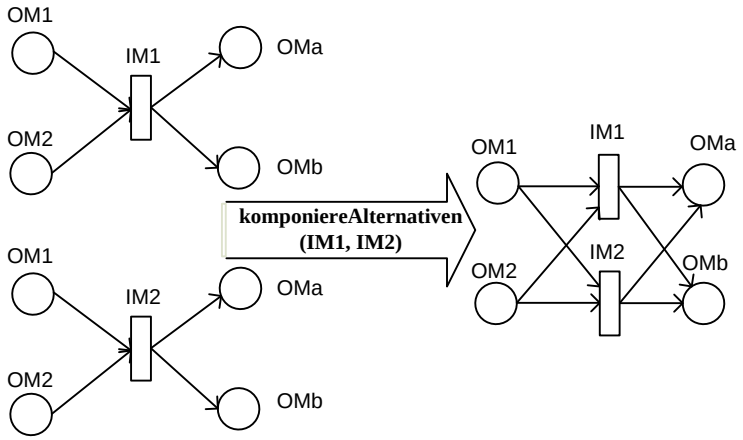


Abbildung 41: Funktion: (BSF/AIN) komponiereAlternativen (IM, IM)

Die Eingangs- und Ausgangsstellen der beiden Transitionen (IM1 und IM2) sind identisch. Eine Komposition ist bereits durch die Verschmelzung der Stellen gegeben, wobei die zu verschmelzenden Stellen eindeutig durch ihre Bezeichner identifiziert sind.

Wenn keine Alternativen angeboten werden sollen, so definiert die Alternative mit der besten Eignung das Ergebnis der Funktion „analysiereAbstrakteInteraktion()“.

Die Funktion **„analysiereAlternative()“** nutzt den OM-Katalog, um eine Liste aller Varianten zu ermitteln, die diese Alternative realisieren. Mit Hilfe der Funktion „analysiereVariante()“ wird eine Realisierung ermittelt, die schließlich mit der Funktion „bewerteBSF/AIN“ bewertet wird. Die beste Lösung definiert das Ergebnis der Funktion.

In der Funktion **„analysiereVariante()“** wird schließlich die Analyse der Realisierungen einer Interaktion durchgeführt. Eine Variante verfügt über Verweise auf Benutzerschnittstellenfragmente und AINs. Beide können mit Hilfe der Funktion „bewerteBSF/AIN()“ bewertet werden.

Die Funktion **„bewerteBSF/AIN()“** ist in der Lage, sowohl ein AIN, als auch ein BSF anhand eines übergebenen Nutzungsprofils zu bewerten, indem sie eine rationale Zahl zwischen „0“ und „9“ zurückliefert. „0“ bedeutet dabei, dass das AIN bzw. BSF ungeeignet ist und „9“ stellt eine optimale Lösung dar. Tatsächlich bewertet werden nur BSF anhand einer Analyse des Nutzungskontextvektors des BSF und des Nutzungsprofils. Hierzu wird die Funktion **„bewerteBSF()“** verwendet, die in Abschnitt 8.2.3 genauer erläutert wird. Die Bewertung eines AIN beruht auf dem Aufruf der Funktion „bildeBSF-AIN()“, die für das zu untersuchende AIN als Ergebnis ein BSF-AIN[] liefert. Aus dieser Datenstruktur können nun die BSF zur Realisierung ermittelt und bewertet werden. Der Durchschnitt der Eignung der verwendeten Benutzerschnittstellenfragmente definiert die Eignung des AINs.

Der beschriebene Algorithmus basiert auf einer Rekursion innerhalb des OM-Kataloges. Untersucht wird die Interaktionsbeschreibung, die durch ein AIN beschrieben wird. Jede Interaktionsmetapher dieses AINs wird analysiert, indem im Objektmetaphern-Katalog nach BSF und AINs gesucht wird, die diese Interaktion beschreiben. Liegt ein AIN vor, so wird dieses mit der gleichen Funktion „bildeBSF-AIN()“ wie die Interaktionsbeschreibung untersucht. Diese rekursive Vorgehensweise birgt die Problemstellung, dass nicht sichergestellt werden kann, dass der Algorithmus auch tatsächlich stoppt. Dieser Fall kann auftreten, wenn sich die AINs im OM-Katalog zyklisch referenzieren. Eine Identifikation solcher Zyklen ist einfach zu realisieren, will man diese Zyklen aber auch auflösen, so bedarf es menschlicher Unterstützung. Um die Funktionsfähigkeit sicherzustellen, wird vorausgesetzt, dass der OM-Katalog frei von solchen Schleifen ist. Damit endet die Rekursion, wenn eine Interaktion erreicht ist, für die kein AIN mehr angegeben werden kann. In diesem Fall muss es laut Definition ein BSF geben, das diese Interaktion realisiert.

Eine weitere Problemstellung dieses Ansatzes tritt auf, wenn für bestimmte Nutzungsprofile keine passenden BSF existieren. Ein konstruiertes Extrembeispiel könnte sein, dass als Endgerät im Nutzugsprofil nur ein „analoges Telefon“ spezifiziert wird, in der „Domain Theory“ aber nur BSF für graphische Oberflächen existieren. Der vorgestellte Algorithmus würde zwar ein BSF-AIN erzeugen, die einzelnen BSF wären aber willkürlich gewählt, so dass die Methode „bewerteBSF()“ für alle BSF „0“ zurückliefern würde, und somit Kriterien für die Auswahl von BSF fehlen.

Dass für bestimmte Nutzungsprofile geeignetere BSF existieren als für andere, kann im Rahmen von bausteinorientierten Ansätzen nicht vermieden werden. Um aber der Problemstellungen zu begegnen, dass „keine Ergebnisse“ erzeugt werden können, wurden drei elementare Interaktionsmetaphern „Eingabe“, „Auswahl“ und „Inhalt“ definiert. Für jede dieser Interaktionsmetaphern existieren entsprechende BSF in der „Domain Theory“ (vgl. Aufbau der „Domain Theory“ in Abschnitt 7.3). Aus jedem dieser BSF kann eine Benutzerschnittstelle für alle Endgeräte, die im Rahmen des Adipositas-Begleiters vorgesehen sind, generiert werden. Möglich ist dies, da diese „elementaren Interaktionsmetaphern“ so „trivial“ sind, dass sie sehr leicht auf den unterschiedlichsten Plattformen umgesetzt werden können. Eine weitere Besonderheit ist, dass die Nutzungskontextvektoren (vgl. Abschnitt 7.3) dieser BSF so aufgebaut sind, dass die Bewertungsfunktion immer einen Wert größer „0“ liefern wird. Das heißt, dass diese BSF genau dann gewählt werden, wenn keine passenden anderen Elemente in der „Domain Theory“ zu finden sind.

Damit der vorgestellt Algorithmus in der Lage ist, diese „Notfalllösung“ aus den elementaren BSF zu ermitteln, muss noch eine weitere Anforderung an den OM-Katalog definiert werden: zu jeder Interaktionsmetapher im OM-Katalog, die nur noch durch ein BSF beschrieben wird, muss zusätzlich ein AIN definiert werden, das nur noch aus den drei elementaren Interaktionsmetaphern besteht.

Dadurch ist sicher gestellt, dass immer eine Lösung ermittelt wird, die sich der elementaren Interaktionsmetaphern bedient. Nicht für jede Interaktionsmetapher muss der zusätzliche Aufwand betrieben werden, dass diese durch AINs bestehend aus elementaren Interaktonsmetaphern beschrieben werden. Vielmehr wird verlangt, dass komplexe Interaktionsmetaphern durch bereits existierende beschrieben werden. Erst diese Beziehungen ermöglichen es, eine Auswahl treffen zu können, die für ein gegebenes Nutzungsprofil optimiert ist.

8.2.3 Bewertung von Benutzerschnittstellenfragmenten anhand eines Nutzungsprofils

Der vorgestellte Algorithmus zur Ermittlung des BSF-AIN, fundiert auf der Möglichkeit, unterschiedliche Benutzerschnittstellenfragmente in Bezug auf ein gegebenes Nutzungsprofil zu bewerten. Die Funktion „bewerteBSF()“ liefert als Ergebnis (im Folgenden Eignung genannt) eine rationale Zahl im Bereich „0“ bis „9“, wobei „0“ als ungeeignet und „9“ als ideal geeignet interpretiert wird.

Grundlage für die Bewertung ist dabei einerseits das Nutzungsprofil, wie es in Abschnitt 6.3 beschrieben wurde und andererseits der Nutzungskontextvektor, der Teil der Beschreibung eines BSF in der “Domain Theory“ ist (vgl. Abschnitt 7.3). Wie in Abschnitt 7.3 beschrieben, sind der Nutzungskontextvektor und das Nutzungsprofil analog zueinander aufgebaut. Basierend auf dem Vergleich zwischen Nutzungsprofil und Nutzungskontextvektor des BSF werden nun Parameter identifiziert, aus denen schließlich die Eignung berechnet wird. Im Anhang C findet sich eine Beschreibung dieses Vorgehens in Form von Pseudocode. Im Folgenden werden das Vorgehen und die Funktionsweise des Verfahrens erläutert.

Es ist vorgesehen, die einzelnen Komponenten des Nutzungsprofils (vgl. Abschnitt 6.3) einzeln zu betrachten und gewichtet in eine Gesamtbewertung einfließen zu lassen. Eine Sonderstellung zur Bestimmung der Eignung nimmt dabei das verfügbare Endgerät ein. Kann ein Benutzerschnittstellenfragment auf einem Endgerät nicht dargestellt werden, so ist dies ein Ausschlusskriterium, auch wenn alle anderen Kriterien positiv sind. Dieser Parameter wird im Folgenden „geräteModifikator“ genannt und wird durch eine rationale Zahl zwischen „0“ und „1“ repräsentiert. Eine „0“ bedeutet dabei, dass das BSF gar nicht auf dem im Nutzungsprofil definierten Endgerät dargestellt werden kann, während eine „1“ eine optimale Realisierung spezifiziert.

Bestimmt wird der Parameter „geräteModifikator“ basierend auf dem UAProf (vgl. Abschnitt 6.3.1) der Bestandteil des Nutzungsprofils ist und zur Beschreibung der Fähigkeiten von Endgeräten eingesetzt wird. UAProf spezifiziert, welche physischen Eigenschaften (Display-Größe, verfügbare Tasten usw.) das Endgerät besitzt und welche Software (z.B. Browser) installiert ist. Ein BSF verfügt über eine definierte

Menge von Präsentationsfragmenten, die im Zuge der Bestimmung des „geräteModifikators“ untersucht werden müssen. Ausgangslage der Fragestellungen ist dabei die Spezifikation der Präsentationsfragmente (vgl. 7.2.1.1), die in Form einer XML-basierten Benutzerschnittstellenbeschreibungssprache[39] vorliegen. Diese Spezifikation wird durch einen Parser interpretiert, der daraus eine ausführbare Benutzerschnittstelle generiert. Der Parser verwendet dabei die Informationen, die im UAProf hinterlegt sind. Die Fragestellung, ob ein BSF auf einem Endgerät realisierbar ist, ist also nicht direkt vom UAProf abhängig, sondern auch vom verfügbaren Parser, der für die Darstellung des BSF auf dem Endgerät verantwortlich ist. Für die hier vorliegende Arbeit wird eine höchst triviale Möglichkeit verwendet, den „geräteModifikator“ automatisch zu ermitteln: Der Parser wird mit der Spezifikation des im Nutzungsprofil beschriebenen UAProf auf alle Präsentationsfragmente des BSF angewendet und überprüft, ob ein Ergebnis produziert wird. Wird für ein Präsentationsfragment kein Ergebnis geliefert oder erfolgt eine Fehlermeldung, so ist der „geräteModifikator“ „0“, ansonsten „1“. Genauere Ergebnisse sind erst durch menschliche Experten oder ausgefeiltere Algorithmen möglich, welche in der Lage sind, die generierten Benutzerschnittstellen zu bewerten. Die Konzeption und Umsetzung von Werkzeugen, die eine automatisierte Durchführung solcher Bewertungen mit Hilfe der Informationen des Nutzungsprofils ermöglichen, stellen an sich schon eine herausfordernde Aufgabe dar, die aber nicht mehr im Fokus dieser Arbeit liegen kann.

Nachdem sichergestellt wurde, dass jedes einzelne Präsentationsfragment dargestellt werden kann, muss in einem zweiten Schritt überprüft werden, ob die gleichzeitige Darstellung mehrer Präsentationsfragmente (falls so im BSF spezifiziert) realisierbar ist. Mit Hilfe des in Anhang D.2 beschriebenen Algorithmus zur Ermittlung eines Interaktionsbaums kann automatisiert aus dem BSF ermittelt werden, welche Präsentationsfragmente zur gleichen Zeit verfügbar sind und somit gleichzeitig auf der Oberfläche dargestellt werden müssen. Das Layoutpattern (vgl. Abschnitt 6.3.5) des Nutzungsprofils definiert schließlich Bereiche, auf denen die Präsentationsfragmente dargestellt werden können. Im Rahmen der Ermittlung des „geräteModifikators“ muss für jede mögliche Kombination gleichzeitig sichtbarer Präsentationsfragmente eine entsprechende Abbildung auf dem Layoutpattern gefunden werden. Kann für eine Kombination solch eine Abbildung nicht ermittelt werden, so muss der „geräteModifikator“ auf „0“ gesetzt werden. Eine eingehende Betrachtung dieser Problemstellung wird in Abschnitt 8.4.1 im Rahmen der Komposition von Benutzerschnittstellenfragmenten durchgeführt.

Neben dem geräteModifikator wird die Eignung in den folgenden Bereichen untersucht, die den Komponenten des Nutzungsprofils (vgl. Abschnitt 6.3) zugeordnet sind:

- Eignung der „Input-Widgets“
  Präferenzen für Interaktionsobjekte;
- Eignung der „Output-Widgets“
  Präferenzen für darstellende Objekte;

[39] In dieser Arbeit wurde XAML als Beschreibungssprache ausgewählt.

- Eignung der „Hardware-Plattform“
  Präferenzen für Interaktionsmedien;
- Eignung des „Layouts“
  Präferenzen für Interaktionsabläufe;

Jede der vier Komponenten wird im Nutzungsprofil basierend auf der Struktur „Parameter, Wert“ repräsentiert. Der Wert (rationale Zahl zwischen „0“ und „9“) im Nutzungsprofil definiert die Präferenz des Anwenders zum Umgang (bzw. zur Nutzung) des im Parameter beschriebenen Elements. Dabei steht eine „9“ für eine sehr gute Akzeptanz, während die „0“ die Nutzung ausschließt.

Der Nutzungskontextvektor verfügt über dieselben Parameter, die aber für die Komponenten-Eignung der „Input-Widgets“, der „Output-Widgets“ und der „Hardware-Plattform“ durch den natürlichen Wert „0“ oder „1“ beschrieben werden. Eine „1“ steht dabei dafür, dass eines der Präsentationsfragmente entsprechende Input- bzw. Output-Widgets beinhaltet, während eine „0“ die Nichtnutzung anzeigt. Die Parametrisierung der Input- und Output-Widgets im Nutzungskontextvektor kann dabei automatisch aus der Spezifikation der Präsentationsfragmente des BSF ermittelt werden.

Die „Hardware-Plattform“ beschreibt die Nutzung physischer Interaktionsmedien (Tastatur, Maus, Sprachesteuerung, Zahlenfeld usw.), die aber nicht direkt dem Präsentationsfragment zu entnehmen sind. Erst durch die Wahl eines Endgerätes kann festgelegt werden, welche Interaktionsmedien benötigt werden, um die Input- und Output-Widgets zu benutzen. Notwendig ist also für jedes Endgerät eine Abbildungsfunktion, die Input- und Output-Widgets den entsprechenden Interaktionsmedien zuordnet. Mit Hilfe solch einer Funktion ist auch hier eine automatische Parametrisierung möglich.

Die Eignung der drei Komponenten (Input-Widgets, Output-Widgets und Hardware-Plattform) wird einfach errechnet, indem der Durchschnitt der Präferenz des Nutzungsprofils für alle Parameter, die im Nutzungskontextvektor mit einer 1 markiert sind, gebildet wird.

Die Parameter der Komponente „Layout“ sind analog zu Input-Widgets, Output-Widgets und der „Hardware-Plattform“ aufgebaut. Hier wird aber sowohl im Nutzungsprofil, als auch im Nutzungskontextvektor die gleiche Semantik verwendet. Dies setzt voraus, dass der Nutzungskontextvektor des BSF entsprechend parametrisiert wurde. Diese Parametrisierung kann nur manuell erfolgen, da hier unter anderem auch Interaktionsflüsse im BSF bewertet werden müssen. Zur Berechnung der Eignung des „Layouts“ wird die Differenz (zwischen Nutzungsprofil und Nutzungskontextvektor) der einzelnen Parameter herangezogen. Durch die einfache Formel „9 – Differenz“ wird die Eignung für jeden Parameter bestimmt. Der Durchschnitt aller Parameter beschreibt auch hier wieder die Eignung der Komponente.

Die Eignung des BSF kann beispielsweise über folgende Formel ermittelt werden:

$$EIGNUNG = \left(\frac{\text{inputWidget}}{4} + \frac{\text{outputWidget}}{4} + \frac{\text{hardwarePlattform}}{4} + \frac{\text{layoutComponent}}{4} \right) * \text{geräteModifikator}$$

Dabei fließt die Eignung jeder der vier Komponenten zu gleichen Teilen in die Bewertung ein. Der geräteModifikator beschreibt hier die Fähigkeiten des Endgerätes, das BSF darzustellen und ermöglicht eine entsprechende Modifikation des Ergebnisses.

Die hier vorgestellte Methode zur Berechnung der Eignung, kann erheblich weiterentwickelt werden, um aussagekräftigere Ergebnisse zu erzielen. So besteht natürlich die Möglichkeit, die vier Komponenten unterschiedlich zu gewichten, oder auch in die Berechnung der einzelnen Komponenten einzugreifen, um differenzierter bewerten zu können. Für den eingeführten Anwendungsfall Adipositas-Begleiter reicht bereits solch eine einfache Lösung aus, um aussagekräftige Ergebnisse zu erhalten und die gesetzte Zielvorgabe zu erreichen.

8.3 Generierung des BSF-AINs am Beispiel des Adipositas-Begleiters

In den vorhergehenden Abschnitten wurde ein Algorithmus beschrieben, mit dessen Hilfe basierend auf einer Interaktionsbeschreibung des Objektmetaphern-Kataloges, der „Domain Theory“ und eines Nutzungsprofils ein BSF-AIN generiert wird. Ziel des vorgestellten Algorithmus ist es, die Interaktionsbeschreibung, die durch ein AIN spezifiziert wurde, so auf implementierungsnahe Benutzerschnittstellenfragmente (BSF) abzubilden, dass das generierte BSF-AIN den im Nutzungsprofil definierten Anforderungen am besten genügt. „Am besten genügt“ heisst, dass das arithmethische Mittel der Bewertungsfunktionen (vgl. Abschnitt 8.2) für die im BSF-AIN definierten BSF (und dem gegebenen Nutzungsprofil) am höchsten ist.

Im Folgenden wird anhand eines Ausschnitts aus dem Anwendungsbeispiel Adipositas-Begleiter präsentiert, wie durch die Wahl geeigneter BSF, durch die Dekompositionen der Interaktionsmetaphern und durch die Auswahl aus Interaktionsalternativen und –varianten, ein in Bezug auf das Nutzungsprofil passendes BSF-AIN generiert wird.

Als Beispiel wurde die in Abbildung 26 (in Abschnitt 5.6) eingeführte Interaktionsbeschreibung herangezogen, in welcher das Anlegen einer Mahlzeit im Umfeld des Adipositas-Begleiters spezifiziert wird.

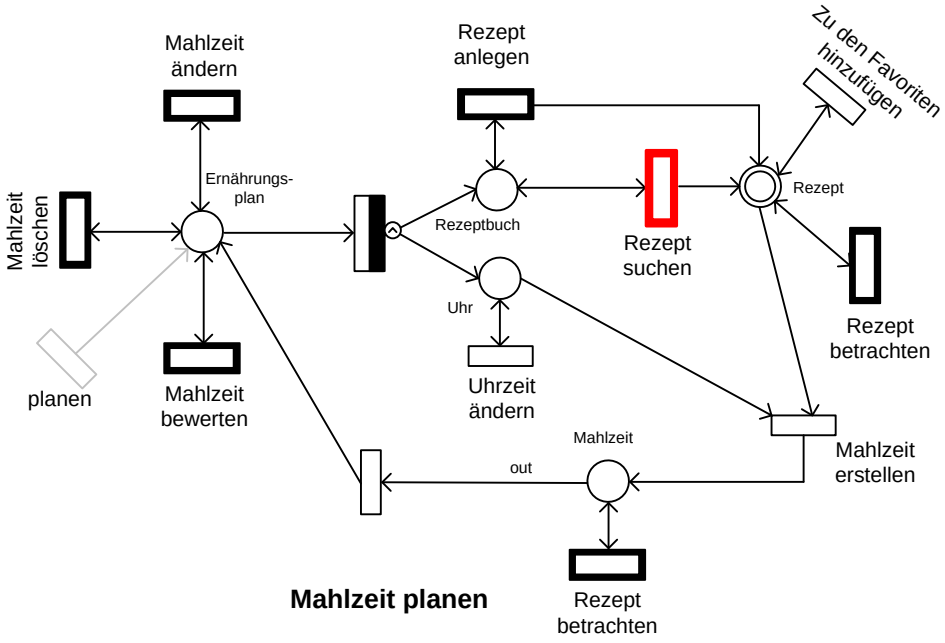


Abbildung 42: Interaktionsmetapher „Rezept Suchen“ im Kontext des Adipositas-Begleiters

Abbildung 42 visualisiert die Interaktionsbeschreibung aus Abschnitt 5.6 (vgl. Abbildung 26). Sie wurde an dieser Stelle nochmals eingefügt, um dem Leser die folgenden Generierungsschritte besser zu veranschaulichen und die Interaktionsmetapher „Rezept suchen“ in den Kontext des Adipositas-Begleiters zu stellen.

Zentraler Ansatzpunkt des Algorithmus zur Generierung des BSF-AINs ist die Dekomposition von Interaktionsmetaphern mit Hilfe des OM-Kataloges. Abbildung 43 visualisiert einen Ausschnitt aus dem OM-Katalog, der die Interaktionsmetapher „Rezept suchen“ umfasst. Aus dem OM-Katalog wird ersichtlich, dass es sich bei der im AIN modellierten Interaktionsmetapher „Rezept suchen“, um eine „Abstrakte Interaktionsmetapher“ handelt, die durch Alternativen und Varianten spezifiziert wird. Wie in Abschnitt 5.6 dargestellt, können Interaktionsbeschreibungen gleichermaßen auch Alternativen oder Varianten zur Modellierung umfassen. Die Modellierung einer „Abstrakten Interaktionsmetapher“ eröffnet dem Algorithmus zur Generierung des BSF-AINs den umfangreichsten Gestaltungsspielraum, da Alternativen, die Komposition von Alternativen und schließlich die unterschiedlichen Varianten für die Ermittlung eines geeigneten BSF herangezogen werden können. Aus dem Ausschnitt des OM-Kataloges in Abbildung 43 lässt sich entnehmen, dass drei Möglichkeiten für die Ermittlung eines BSF für die gegebene Interaktionsmetapher in Frage kommen. Dies sind die beiden AINs „SucheImKatalog_AIN“ und „EinfacheSucheInKategorien_AIN“ und schließlich das BSF „KatalogSuche.bsf“. Dabei ist zu bedenken, dass nur BSF bewertet werden können. Für die Bewertung eines AINs ist es folglich erforderlich, ein geeignetes BSF-AIN zu ermitteln. Somit ist der rekursive Aufbau des Algorithmus zu erklären, der ausgehend von einer Interaktionsmetapher unterschiedliche Varianten erzeugen muss, um diese schließlich in Bezug zu einem Nutzungsprofil bewerten zu können.

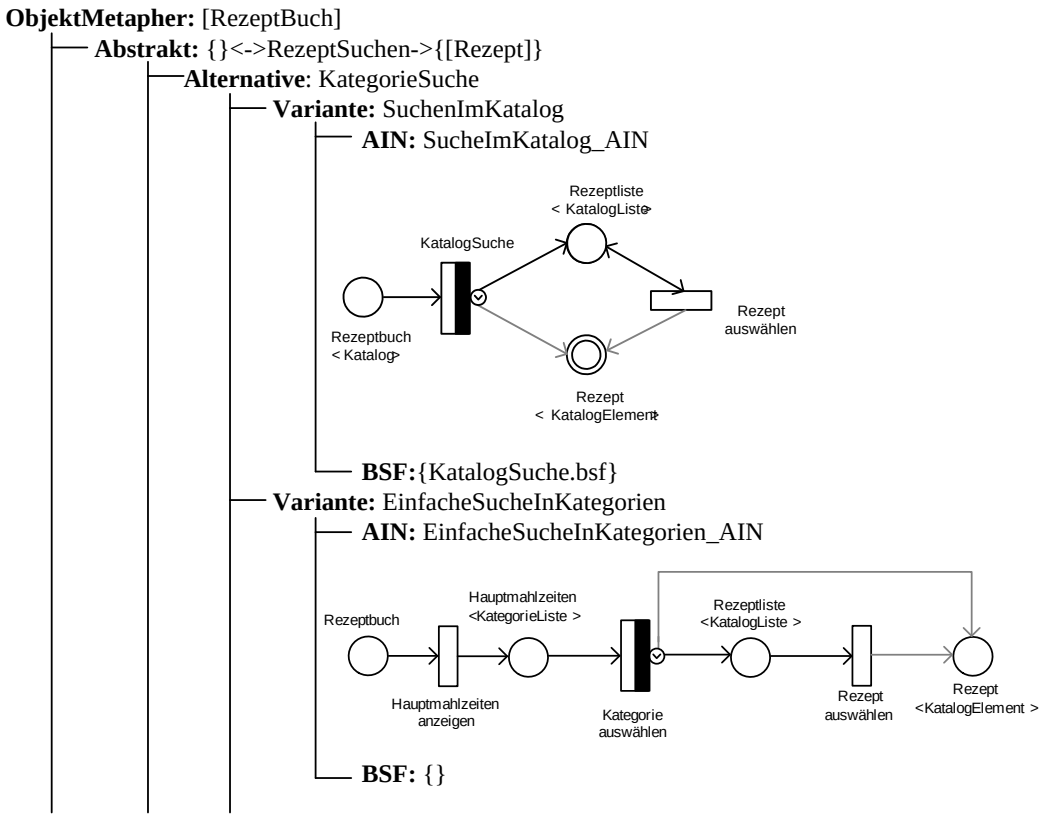


Abbildung 43: „RezeptSuchen“ Ausschnitt aus dem OM-Katalog

Abbildung 44 veranschaulicht vereinfacht diese Funktionsweise des Algorithmus zur Generierung des BSF-AINs anhand des Beispiels „Rezept suchen“. Ausgehend von der Interaktionsmetapher „Rezept suchen“ werden in einem ersten Schritt die drei oben beschriebenen Möglichkeiten zur Realisierung der Interaktionsmetapher dargestellt. Dabei kann nur das BSF „KatalogSuche.bsf“ direkt im ersten Schritt bewertet werden, die AINs müssen weitergehend analysiert werden. Als Beispiel für solch eine Analyse wurde hierzu das AIN „SucheImKatalog_AIN“ genauer betrachtet. Es definiert ein AIN mit zwei Interaktionsmetaphern „Katalogsuche“ und „Rezept auswählen“. Um dieses AIN bewerten zu können, müssen für die beiden Interaktionsmetaphern passende BSF identifiziert werden. Hierzu wird wiederum der OM-Katalog herangezogen. Wie dem OM-Katalog im Anhang A zu entnehmen ist, handelt es sich bei der Interaktionsmetapher „KatalogSuche“ ebenfalls um eine „Abstrakte Interaktion“, die aber drei Alternativen für deren Realisierung vorsieht. An dieser Stelle kommt der Teil des Algorithmus „komponiereAlternativen()“ zum Einsatz, der mögliche Interaktionsalternativen im Interaktionsfluss vorsieht. Dabei wird das AIN erweitert, indem dem Nutzer mehrere Alternativen zur Durchführung einer bestimmten Interaktion angeboten werden.

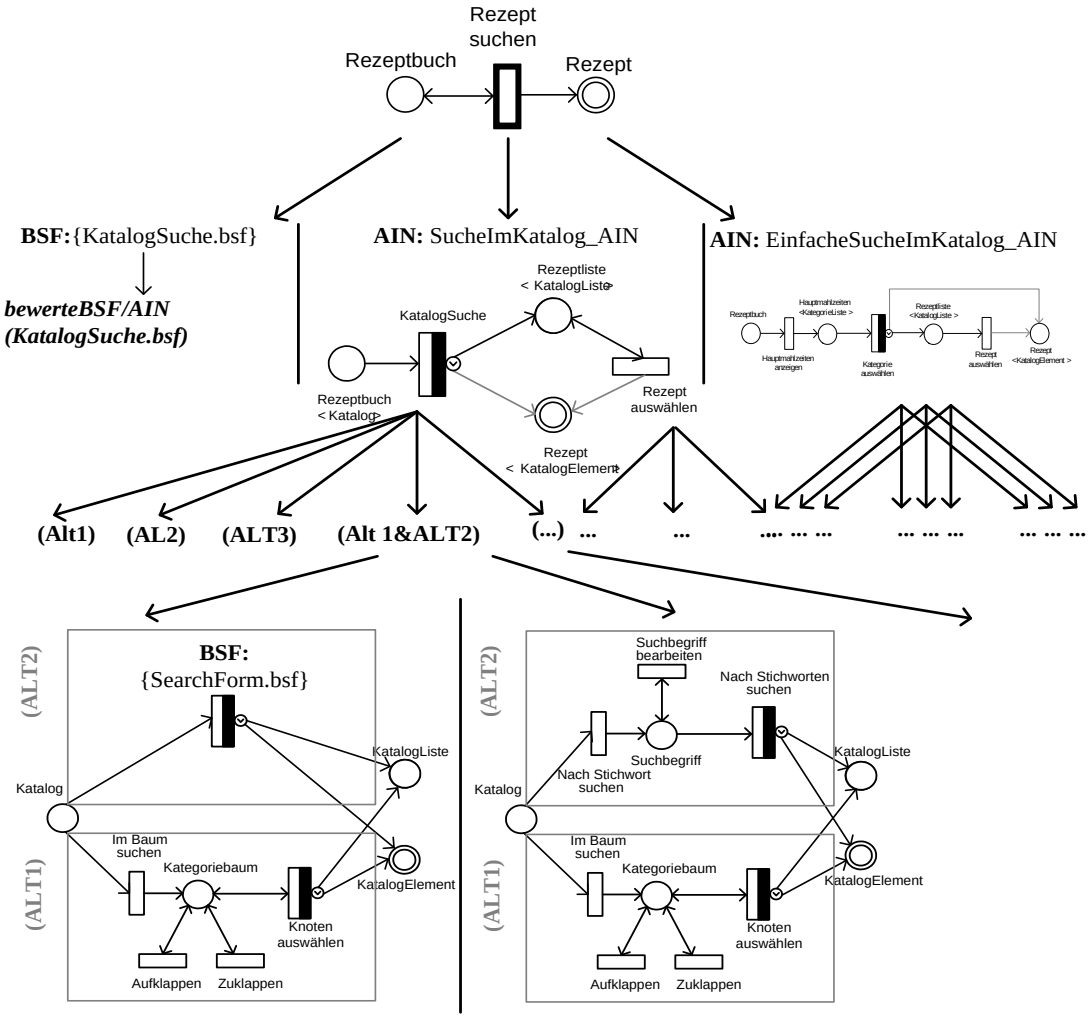


Abbildung 44: Abbildung IM->BSF-AIN am Beispiel „Rezept suchen“

In Abbildung 44 ist dargestellt, welche Möglichkeiten sich für die Komposition der Alternativen ergeben: dies sind enweder eine einzelne Alternative „Alt1“, „Alt2“, „Alt3“, oder aber die Komposition unterschiedlicher Alternativen. „Alt1 & Alt2“ steht dabei stellvertretend für alle Teilmengen aus {Alt1, Alt2 und Alt3}, die aus mehr als einem Element bestehen. Diese Teilmengen werden mit der Methode „komponiereAlternativen()“ zu einem neuen AIN verschmolzen. Abbildung 44 stellt exemplarisch die Komposition der Alternativen „(Alt1 & Alt2)“ dar. Dabei entsteht eine Menge von AINs, da jede Alternative durch unterschiedliche Varianten (oder aber auch BSF) realisiert werden kann. Zwei mögliche Alternativen (vgl. Alternativen der Abstrakten Interaktion „Katalogsuche“ im OM-Katalog in Anhang A) sind im unteren Teil der Abbildung 44 dargestellt.

Das in Abbildung 44 präsentierte Beispiel für die Arbeitsweise des Algorithmus verdeutlicht, dass der Algorithmus eine Dekomposition des zu bewertenden AINs durchführt, indem jede Interaktionsmetapher durch unterschiedliche Varianten (AINs, BSF) beschrieben wird. Das durch diese Dekomposition entstan-

dene AIN wird mit jedem weiteren Schritt weiter zerlegt, bis nur noch eine Repräsentation durch BSF entsteht, die schließlich bewertet werden kann.

Anhand der drei in Abschnitt 6.4.4 beschriebenen Nutzungsprofile („Best Ager zuhause am Fernseher“, „Youngster unterwegs mit dem PDA“ und „Mid Ager bei der Arbeit am PC“) und unter Nutzung der in Abschnitt 7.3 eingeführten Domain-Theory (Ausschnitt dargestellt in Tabelle 10) und des in Abschnitt 5.5 definierten OM-Kataloges (Ausschnitt in Tabelle 2) kann die Generierung des BSF-AIN durchgeführt werden.

Ein Ausschnitt der Generierungsergebnisse wird in den folgenden Abbildungen (Abbildung 45, Abbildung 46 und Abbildung 47) dargestellt. Gegenübergestellt wird die Analyse der Interaktionsmetapher „Rezept suchen“, die auf der Objektmetapher Rezeptbuch ausgeführt wird (vgl. Interaktionsbeschreibung in Abbildung 42).

Jedes der drei AIN (Abbildung 45, Abbildung 46 und Abbildung 47) repräsentiert eine Realisierung der Interaktionsmetapher „Rezept Suchen“ für das jeweilige Nutzungsprofil. Für die Darstellung der BSF-AINs wurde die AIN-Notation beibehalten, wobei jeder Interaktionsmetapher eindeutig ein BSF zugeordnet werden kann. Um den Dekompositionsprozess graphisch darzustellen, wurden Rechtecke in unterschiedlich dunklen Grüntönen verwendet. Dabei gilt, dass hellere Rechtecke durch Dekomposition des umfassenden dunkleren Rechtecks generiert worden sind. Die Darstellung des „dekomponierten“ AINs wurde aufgenommen, um darzustellen, welche Verfeinerung der Interaktionsmetapher schließlich als bestes Ergebnis (definiert durch die Bewertungsfunktion) für das gegebene Nutzungsprofil identifiziert wurde.

Die Benennung der Benutzerschnittstellenfragmente wurde hier zur Vereinfachung mit den Interaktionsmetaphern gleichgesetzt.

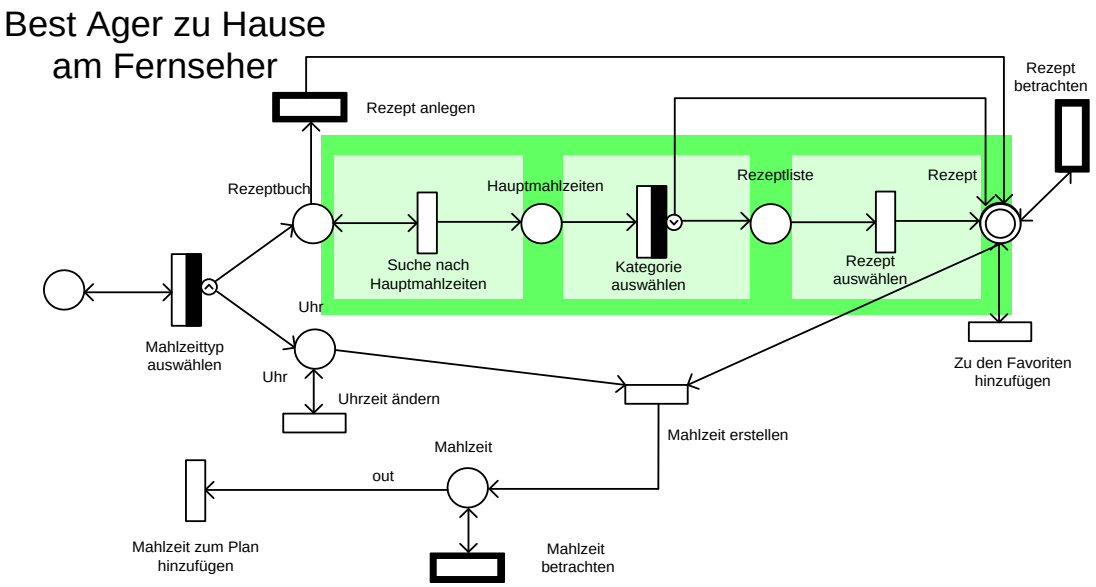


Abbildung 45: BSF-AIN für das Nutzungsprofil „Best Ager zuhause am Fernseher“

Einen sehr einfachen und sequentiellen Ablauf zeigt das AIN in Abbildung 45 für das Nutzungsprofil „Best Ager zuhause am Fernseher“. Das Resultat begründet sich einerseits in den Einschränkungen des Endgerätes (Set-Top-Box und Fernseher), andererseits aber auch durch die Charakterisierung des Nutzers als „nicht technikaffin“. Obwohl das Display des Fernsehers sehr groß ist, muss berücksichtigt werden, dass die Entfernung zum Betrachter ebenfalls groß ist. Somit ist die Menge an Informationen und auch Interaktionsobjekten, die auf einem Bildschirm angeboten werden, erheblich geringer als auf einem herkömmlichen PC-Display. BSF, welche die gleichzeitige Präsentation von unterschiedlichen Präsentationsfragmenten vorsehen, können somit gar nicht dargestellt werden. Verschärft wird diese Anforderung weiterhin dadurch, dass dem Benutzer laut Nutzungskontext (vgl. Tabelle 8) auch ein mäßiges Sehvermögen zugesprochen wird. Für „nicht technikaffine“ Benutzer sollten weiterhin parallele Interaktionsabläufe, Varianten und Alternativen vermieden werden, um einen möglichst „geführten“ Interaktionsablauf gewährleisten zu können. Schließlich wurde als Realisierung der „Rezept suchen“ eine Kategorie-basierte Variante gewählt. Diese erfordert zwar mehrere Interaktionsschritte, verzichtet aber weitgehend auf komplexe Interaktionen (Texteingabe, Menüs usw.). Auch diese Auswahl begründet sich auf dem Nutzungskontext, in dem die motorischen Fähigkeiten des Nutzers als schlecht definiert wurden.

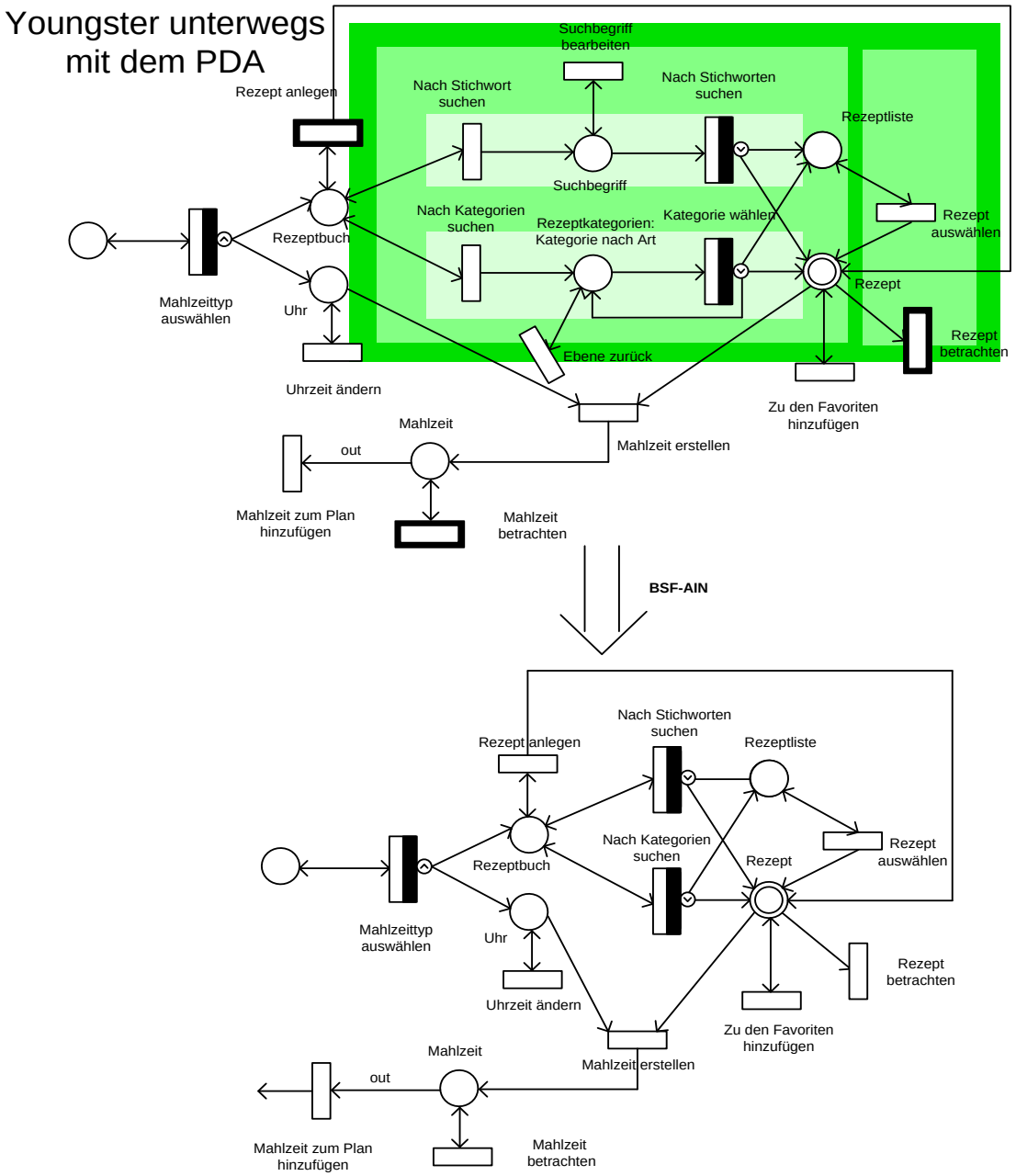


Abbildung 46: BSF-AIN für das Nutzungsprofil „Youngster unterwegs mit dem PDA“

Das Ergebnis der Generierung für das Profil „Youngster unterwegs mit dem PDA“ wird in Abbildung 46 präsentiert. Im Gegensatz zur vorherigen Abbildung (Abbildung 45) zeichnet es sich durch das Angebot von zwei Alternativen („Suche nach Stichworten“, „Suche nach Kategorien“) für die Durchführung der Suche aus. Da auch für die Nutzung eines PDAs ähnliche Einschränkungen in Bezug auf die Bildschirmgröße und die Menge der darzustellenden Informationen gelten, sind auch hier Benutzerschnittstellenfragmente heran zu ziehen, die einfach strukturiert sind. Bedingt durch die Beschreibung des Nutzers als „technikaffin“ im Nutzungskontext sollte aber die Interaktionsdurchführung möglichst flexibel und vari-

antenreich sein. Alternativen stellen einen Mechanismus dar, Varianten anbieten zu können, ohne komplexe Benutzerschnittstellenfragmente benutzen zu müssen.

Abbildung 46 beinhaltet weiterhin ein zweites Netz (oben und unten in Abbildung 46). Das obere Netz ist analog zur Abbildung 45 aufgebaut und zeigt die mögliche Dekomposition der Benutzerschnittstellenfragmente. Das untere Netz beschreibt dagegen die durch den Algorithmus ausgewählten BSF. Aus der Abbildung ist ersichtlich, dass die in der unteren Abbildung dargestellten Benutzerschnittstellenfragmente „Nach Stichworten suchen“ und „Nach Kategorien suchen“ die inneren (hellgrün dargestellten) Subnetze realisieren.

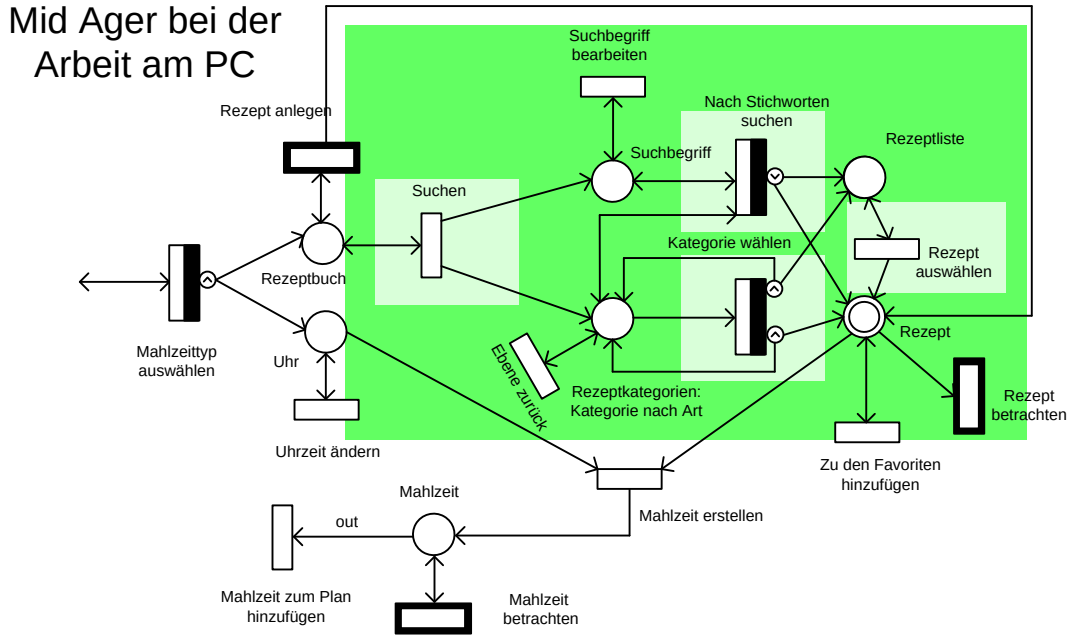


Abbildung 47: BSF-AIN für das Nutzungsprofil „Mid Ager bei der Arbeit am PC“

Abbildung 47 stellt schließlich die „Rezept Suche“ für den Nutzungskontext „Mid Ager bei der Arbeit am PC“ dar. Das Ergebnis der Generierung zeigt ein komplexes Netz, in welchem parallel auf unterschiedliche Suchmethoden zurückgegriffen werden kann. Begründet ist diese Auswahl der Benutzerschnittstellenfragmente durch das verwendete Endgerät (einen Desktop-PC), genauer gesagt durch die Darstellungsmöglichkeiten und auch Interaktionsmöglichkeiten des Gerätes. Weiterhin zeichnet sich der im Nutzungskontext definierte Benutzer durch den Wunsch nach einer effizienten Interaktionsdurchführung aus und besitzt auch die Fähigkeit, komplexe Interaktionsobjekte beherrschen zu können. Somit können auch komplexe Benutzerschnittstellenfragmente herangezogen werden, die auch die notwendige Flexibilität und Effizienz in der Interaktionsdurchführung sicherstellen.

Eine detaillierte Beschreibung dieses Generierungsschrittes findet sich in Kapitel 9 dieser Arbeit.

8.4 Komposition der Benutzerschnittstellenfragmente

Die Komposition der Benutzerschnittstellenfragmente verfolgt als finaler Prozessschritt im Generierungsprozess die Zielsetzung, die im Vorfeld identifizierten Bausteine zu einer vollständigen Benutzerschnittstelle zusammenzufügen. Das BSF-AIN beschreibt dabei einerseits die identifizierten Bausteine (die Benutzerschnittstellenfragmente) und liefert andererseits auch Informationen über die Beziehungen zwischen diesen. Letztere sind in Form eines AIN modelliert, das dem BSF-AIN zugrunde liegt.

In einem ersten Schritt wird die Aufgabenstellung der Komposition detaillierter betrachtet und Problemstellungen, die mit der Komposition einhergehen, konkretisiert. Die primäre Ursache für diese Problemstellungen liegt in dem Vorgehen des Algorithmus zur Erstellung des BSF-AINs, das vorsieht, dass jedes BSF einzeln, nicht aber das Zusammenspiel der einzelnen BSF betrachtet wird. Basierend auf diesen Betrachtungen wird ein Algorithmus skizziert, der automatisiert aus einem gegebnen BSF-AIN und dem Nutzungsprofil eine ausführbare Benutzerschnittstelle generiert. Die dafür benötigten Methoden und Werkzeuge werden in den darauf folgenden Abschnitten bereitgestellt.

Der entwickelte Algorithmus basiert dabei auf einer Analyse aller Interaktionspfade des BSF-AINs. Ein Interaktionspfad beschreibt eine Möglichkeit für die Durchführung des BSF-AIN in Form einer Sequenz von Interaktionsmetaphern. Mit Hilfe der Menge aller möglichen Sequenzen können alle validen Zustände vollständig beschrieben werden, die das BSF-AIN annehmen kann. Mit Hilfe dieser Interaktionspfade ist es möglich, das BSF-AIN zu untersuchen, und auch Aussagen zu treffen, ob eine Komposition unter Berücksichtigung des Nutzungsprofils möglich ist. Ist eine Komposition nicht durchführbar, muss das BSF-AIN modifiziert werden. Die Modifikationen müssen dabei so gewählt werden, dass möglichst alle im BSF-AIN modellierten Beziehungen zwischen den BSF erhalten bleiben. Im Rahmen dieser Arbeit werden Methoden bereitgestellt, die automatisiert die notwendigen Modifikationen am BSF-AIN vornehmen.

Als Ergebnis liegt ein (gegebenenfalls modifiziertes) BSF-AIN vor. Die Komposition erfolgt auf Basis einer „Intervallalgebra“, mit der sich die Beziehungen zwischen den BSF durch „Relationen der Intervallalgebra“ beschreiben lassen. In einem ersten Schritt werden alle Beziehungen zwischen den BSF, die im BSF-AIN modelliert sind, durch „Relationen der Intervallalgebra“ abgebildet. In einem zweiten Schritt wird nun auf Basis dieser Beschreibung komponiert, indem für jede Relation Methoden bereitgestellt werden, welche die Komposition basierend auf der formalen Beschreibung des BSF realisieren.

Die Funktionsweise des Algorithmus und der für die Durchführung notwendigen Methoden werden am Beispiel des Adipositas-Begleiters exemplarisch erläutert.

8.4.1 Zielsetzung bei der Komposition von Benutzerschnittstellenfragmenten

Um die Aufgabenstellung der Komposition greifbar zu machen, muss in einem ersten Schritt diskutiert werden, was die Komposition von BSF tatsächlich bedeutet und wie sie in realen Benutzerschnittstellen umgesetzt wird. Die Realisierung der Komposition kann auf unterschiedliche Weise erfolgen. So kann man bereits von Komposition sprechen, wenn zwei BSF zur gleichen Zeit für den Benutzer sichtbar sind, auch wenn sie sich gegenseitig überhaupt nicht beeinflussen können. Diese Art der Komposition wird häufig in Web-Portalen verwendet und wird dort beispielsweise durch „Portlets“ (vgl. [Kuss05]) realisiert. Ein Portlet kapselt die Benutzerschnittstelle für eine eigenständige Anwendung, die in ein bestehendes Web-Portal integriert wird, indem ein kleiner Ausschnitt der Oberfläche ausschließlich für das Portlet reserviert wird. In diesem Teil der Benutzerschnittstelle wird eine eigenständige Anwendung präsentiert. Dabei ist diese Anwendung zumeist unabhängig von dem restlichen Web-Portal und (falls vorhanden) anderen Portlets. Auch wenn diese Art der Komposition trivial erscheinen mag und keinen Einfluss auf Abläufe zu einer Aufgabenerfüllung hat, ist sie aus Sicht der Benutzerinteraktion dennoch von Bedeutung, da jedes Portlet die Interaktionsmöglichkeiten erweitert. Neuere Ansätze von Portlets (vgl. [Kuss05]) ermöglichen auch einen Zugriff auf Portalfunktionen, die dann aber manuell zu kodieren (programmieren) sind. Die Bereitstellung dieser Portalfunktionen erlaubt es, dass Informationen eines Portlets an ein anderes Portlet (oder andere Anwendungen auf dem Web-Portal) weitergeleitet werden können. Somit haben Interaktionen eines Nutzers mit einem Portlet direkt Auswirkungen auf ein anderes, was ebenfalls als eine Art der Komposition zu verstehen ist. BSF kommen dem erweiterten Verständnis der Portlets sehr nahe, wobei Portlets in ihrer Spezifikation ausschließlich auf Web-Schnittstellen beschränkt sind.

Die sequentielle Abfolge von Benutzerschnittstellenfragmenten ist eine vereinfachte Variante der Komposition von BSF, bei denen ein BSF Auswirkungen auf ein anderes hat. Sequentiell bedeutet hier, dass durch das Beenden eines BSF ein anderes gestartet wird. Diese Art der Komposition findet sich bereits indirekt in der Definition (vgl. Abschnitt 7.2.1) von BSF wieder. Ein BSF ist demnach durch einen Interaktionsablauf definiert und verfügt somit über definierte Start- und auch Endpunkte.

Die Kompositionsmöglichkeiten, die auf Interaktionsabläufen beruhen, sind natürlich nicht auf Sequenzen beschränkt. Jede Interaktion innerhalb eines BSF kann auch Auswirkungen auf andere BSF haben, ohne dass dieses direkt beendet wird. Auch BSF die parallel starten und auch abgearbeitet werden müssen, beschreiben Kompositionsmöglichkeiten, wobei sich parallele BSF nicht notwendigerweise beeinflussen müssen. Dabei handelt es sich keinesfalls um theoretische Konstrukte. Benutzerschnittstellen bedienen sich oftmals paralleler Fenster oder Frames, die sich gegenseitig beeinflussen können. Dieser Betrachtung nach liegt es nahe die Komposition von BSF als Komposition von Interaktionsabläufen zu verstehen.

Wie die BSF zu komponieren sind, wird im BSF-AIN modelliert, wobei für die Modellierung Konstrukte des AIN eingesetzt werden. AINs nutzen die Sichtbarkeit von Objektmetaphern und die damit verbundenen Verfügbarkeiten von Interaktionsmetaphern, um Alternativen, parallele Abläufe und Bedingungen im Interaktionsprozess zu modellieren. Betrachtet man das BSF-AIN aus Abbildung 46, das den Interaktionsablauf zum „Erstellen einer Mahlzeit“ im Rahmen des Adipositas-Begleiters für den Nutzungskontext „Youngster unterwegs mit dem PDA“ beschreibt, so finden sich auch hier alle Elemente eines AINs wieder. Lediglich die Interpretation dieser Elemente ist eine andere. So repräsentieren Transitionen die eigentlichen BSF, während die Stellen Präsentationsfragmente des zugehörigen BSF modellieren (vgl. Abschnitt 8.3). Die Kanten modellieren schließlich die Beziehungen zwischen den BSF und geben Auskunft, welchen Bedingungen die Komposition genügen muss.

Bevor die Komposition eingehender betrachtet wird, ist es notwendig, eine Besonderheit der generierten BSF-AINs herauszuarbeiten. Wie man den BSF-AINs in Abbildung 46 und Abbildung 47 entnehmen kann, ist es auffällig, dass das Konstrukt der Nebenstellen[40] äußerst häufig eingesetzt wird. So ist das „Rezeptbuch“ in dem BSF-AINs eine Nebenstelle für alle anderen Transitionen. Das bedeutet aber, dass diese Stelle durchgehend markiert bleibt, sobald sie einmal markiert wurde. Die Interaktion „Rezept suchen“ (in Abbildung 47) markiert davon ausgehend gleichzeitig zwei weitere Stellen, die selbst wieder größtenteils Nebenstellen sind. Modelliert wird dadurch, dass der Benutzer möglichst frei in seiner Interaktionsdurchführung ist, und Interaktionen wiederholen, oder begonnene Interaktionen zu einem anderen Zeitpunkt wieder aufnehmen kann. Begründet liegt die in den BSF-AIN (in Abbildung 46 und Abbildung 47) modellierte Flexibilität im zugrunde liegenden Nutzungsprofil, das eine flexible Interaktionsdurchführung gewährleisten soll. Ist solch eine Flexibilität nicht gewünscht (vgl. das BSF-AIN in Abbildung 45 für das Nutzungsprofil „Best Ager zuhause am Fernseher“), so spiegelt sich dies auch direkt im BSF-AIN wieder (Vermeidung von Nebenstellen und parallelen Abläufen).

Der modellierten Flexibilität in der Interaktionsdurchführung stehen Einschränkungen gegenüber. Beispielsweise verfügen graphische Benutzerschnittstellen nur über eine bestimmte Fläche zur Präsentation von Fenstern, Interaktionsobjekten und Inhalten, die durch die Bildschirmgröße und die Auflösung beschränkt ist. Aber auch andere Interaktionsformen unterliegen solchen Restriktionen, so erlaubt beispielsweise der klassische Telefondialog nur strikt sequentielle Abläufe.

Es besteht also die Notwendigkeit, die modellierte Flexibilität in der Interaktionsdurchführung an die Rahmenbedingungen der Darstellungsmöglichkeiten anzupassen. Hierzu müssen Restriktionen eingeführt werden, welche die Flexibilität der Interaktionsdurchführung einschränken und auf das für die gegebene Benutzerschnittstelle Mögliche reduzieren.

---

[40] Nebenstellen sind Stellen im AIN, die für eine Transition zugleich Eingangs- als auch Ausgangsstelle sind (vgl. Abschnitt 5.3.1).

Auch wenn bereits zum Aufbau des BSF-AINs das Nutzungsprofil herangezogen und Bausteine ausgewählt wurden, welche den Anforderungen bestmöglich gerecht wurden, kann die Komposition nicht auf das reine Verbinden dieser Bausteine reduziert werden. Ziel der Komposition ist es, das Zusammenspiel der einzelnen Bausteine aufeinander abzustimmen. Zu berücksichtigen sind dabei die zur Verfügung stehenden Darstellungsmöglichkeiten des verwendeten Endgerätes, die durch das Layoutpattern (vgl. 6.3.5) des Nutzungsprofils weiter strukturiert sind. Daraus sind Restriktionen abzuleiten, welche dazu benutzt werden können, die im BSF-AIN modellierten Varianten in der Interaktionsdurchführung für eine Realisierung der Benutzerschnittstelle zu konkretisieren.

Im Rahmen der vorliegenden Arbeit wurde die beschriebene Problemstellung konkretisiert und zusätzliche Einschränkungen getroffen. So liegt mit dem Layoutpattern (vgl. Abschnitt 6.3.5) des Nutzungsprofils eine konkrete Beschreibung der darstellbaren Bereiche vor, welche die Möglichkeiten zur Darstellung bereits reglementiert.

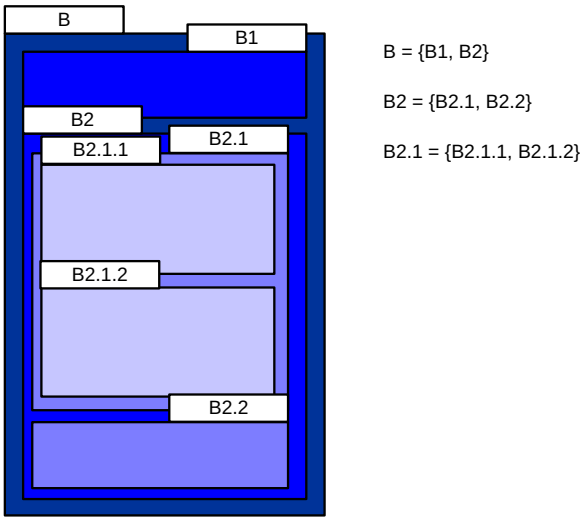


Abbildung 48: Layoutpattern für einen PDA

Abbildung 48 visualisiert ein Layoutpattern, wie es für einen PDA definiert werden kann. Die gesamte Oberfläche ist demnach in unterschiedliche Bereiche strukturiert, die wiederum hierarchisch gegliedert sind. So definiert „B“ die gesamte Oberfläche, die in B1 und B2 gegliedert ist. Rechts in Abbildung 48 ist diese Strukturierung nochmals explizit dargestellt, wobei sich die Beschriftung an den Bereichen orientiert, die in den Karteireitern benutzt wurde.

Jeder der Bereiche des Layoutpatterns kann als Container verstanden werden, in welchen eine Darstellung von BSF realisiert wird. Diese Strukturierung vereinfacht die Analyse, wie BSF auf der Benutzerschnittstelle zu positionieren sind. Neben der sonst sehr problematisch zu bewältigenden freien Positionierung ist erschwerend zu berücksichtigen, dass die Darstellung von Präsentationsfragmenten stark variieren kann. Ein Parser generiert aus einer XAML-Beschreibung die tatsächliche Realisierung der Benutzer-

schnittstelle, wobei durch Parametrisierung des Parsers (z.B. auch Vorgaben des zur Verfügung stehenden Raumes auf der Benutzerschnittstelle) unterschiedliche Darstellungen generiert werden können.

Im Rahmen der Bewertung von Benutzerschnittstellenfragmenten anhand eines Nutzungsprofils (vgl. Abschnitt 8.2.3) wurde bereits diskutiert, dass für die Ausführung Präsentationsfragmente auf Bereichen des Layoutpatterns abgebildet werden müssen. BSF, die eine Menge von Präsentationsfragmenten enthalten, können demnach auch eine Menge von Bereichen allokieren, falls mehrere Präsentationsfragmente gleichzeitig sichtbar sein sollen. Dabei ist aber festzustellen, dass nicht notwendigerweise alle diese Bereiche während der gesamten Ausführung des Benutzerschnittstellenfragmentes belegt sein müssen. Um die Komposition von Benutzerschnittstellenfragmenten zu vereinfachen, kann aber angenommen werden, dass ein Benutzerschnittstellenfragment alle benötigten Bereiche durchgehend belegt. Begründet wird diese Restriktion durch die semantische Beziehung zwischen Interaktionen und Benutzerschnittstellenfragmenten. Benutzerschnittstellenfragmente verstehen sich als Realisierungen für die Durchführung von Interaktionen. Folglich bildet ein Benutzerschnittstellenfragment aus der Perspektive des Anwenders eine zusammengehörige Einheit, die sich auch in der graphischen Repräsentation wieder finden sollte.

Dieser Argumentation folgend wird weiterhin angenommen, dass die Menge der Bereiche nicht variiert. Dies ist ebenfalls als Restriktion aufzufassen, da die Darstellung von Präsentationsfragmenten prinzipiell in jedem ausreichend großen Bereich realisiert werden kann. Wie in Abschnitt 8.2.3 dargestellt, ist die Ermittlung geeigneter Bereiche durch einfache Algorithmen zu realisieren. Problematisch dabei ist jedoch die Wahl einer für den Nutzer optimalen Lösung. Hierzu müssen Methoden eingesetzt werden, welche die Bewertung generierter Benutzerschnittstellen ermöglichen, um die einzelnen Lösungen miteinander vergleichen zu können. Im Rahmen dieser Arbeit kann aber insoweit ein pragmatischer Ansatz gewählt werden, dass eine automatisch generierte Lösung angeboten wird, die gegebenenfalls im Arbeitsschritt „Evaluation und Konfiguration“ (vgl. Abschnitt 8.1) des Kompositionsprozesses durch einen menschlichen Experten modifiziert werden kann. Im Rahmen des Aufbaus eines Baukastens für die Benutzerschnittstellengenerierung werden die vorgenommenen Modifikationen gespeichert und wieder verwendet.

Zusammenfassend werden für die Komposition folgende Annahmen festgehalten:

1. Der sichtbare Bereich der Benutzerschnittstelle ist hierarchisch strukturiert und wird durch das Layoutpattern eindeutig beschrieben.
2. Jeder Bereich des Layoutpatterns kann für die Darstellung eines Präsentationsfragmentes genutzt werden. Die Zuordnung von Präsentationsfragmenten zu Bereichen des Layoutpatterns ist statisch. Dass heisst, das jedes Präsentationsfragment genau einem Bereich eines Layoutpatterns zugeordnet ist.
3. Jedes Benutzerschnittstellenfragment belegt für die Dauer seiner Ausführung eine definierte Menge dieser Bereiche des Layoutpatterns.
4. Nach Beendigung des Benutzerschnittstellenfragmentes stehen diese Bereiche wieder zur Verfügung.

Basierend auf diesen Annahmen kann die Problemstellung der begrenzten Oberfläche für die Komposition folgendermaßen formuliert werden: Im BSF-AIN modellierte parallele Abläufe von Benutzerschnittstellenfragmenten können nur dann realisiert werden, wenn keine Überschneidungen in diesen Benutzerschnittstellenfragmenten vorliegen. Dabei spricht man von Überschneidungen, wenn die Schnittmenge der Bereiche des Layoutpatterns der Benutzerschnittstellenfragmente ungleich der leeren Menge ist.

Tabelle 11 visualisiert diese Problemstellung am Beispiel des Adipositas-Begleiter. Dem Beispiel liegt das BSF-AIN für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ (vgl. Abbildung 46) und das in Abbildung 48 dargestellte Layoutpattern für einen PDA zugrunde.

Benutzerschnittstellenfragmente	B1	B2		
		B2.1		B2.2
		B2.1.1	B2.1.2	
„Rezept anlegen“				
„Nach Stichworten suchen“				
„Nach Kategorien suchen“				
„Rezept auswählen“				
„Uhrzeit ändern“				
„Mahlzeit erstellen“				
„Mahlzeit betrachten“				
„Mahlzeit hinzufügen“				
„Zu den Favoriten hinzufügen“				

Tabelle 11: Mapping BSF -> Layoutpattern

Tabelle 11 visualisiert eine Abbildung von BSF auf Bereiche des Layoutpatterns. Dunkel dargestellte Elemente der Tabelle beschreiben Bereiche, die zur Ausführung durch das Benutzerschnittstellenfragment belegt werden, während weiße Bereiche eine Nichtnutzung anzeigen.

Anhand der Tabelle ist deutlich ersichtlich, dass die Ausführung von Benutzerschnittstellenfragmenten durch die zur Verfügung stehenden Bereiche stark reglementiert sein kann. Die Nutzung eines PDAs, wie sie im Beispiel gewählt wurde, verschärft diese Problemstellung durch das kleine Display. Zieht man das BSF-AIN aus Abbildung 46 heran, so ist direkt offensichtlich, dass sich beispielsweise die modellierte Parallelität der Benutzerschnittstellenfragmente „Rezept anlegen“ und „Uhrzeit ändern“ nicht auf das vorgegebene Layoutpattern abbilden lässt.

Ausgangslage für die Komposition muss die im BSF-AIN beschriebene Verknüpfung der Interaktionen sein, wobei Parallelität, falls deren Darstellung nicht möglich ist, in Sequenzen überführt werden muss. Schließlich ist bei den erzeugten Sequenzen zu überprüfen, ob die so erzeugten Abläufe weiterhin die notwendigen Datenflüsse zwischen den Benutzerschnittstellenfragmenten gewährleisten.

8.4.2 Algorithmus zur Komposition von Benutzerschnittstellenfragmenten

Die Komposition der Benutzerschnittstellenfragmente fundiert auf dem BSF-AIN, das einerseits die notwendigen Benutzerschnittstellenfragmente benennt, andererseits aber auch das Zusammenspiel der Benutzerschnittstellenfragmente definiert. Bereits durch den Generierungsprozess des BSF-AINs wird sichergestellt, dass nur Benutzerschnittstellenfragmente ausgewählt werden, die auch über das Layoutpattern darstellbar sind (vgl. Bewertungsfunktion in Abschnitt 8.2.3). Die Betrachtung der Benutzerschnittstellenfragmente erfolgt aber isoliert voneinander. Isoliert bedeutet, dass jedes Benutzerschnittstellenfragment einzeln bewertet wird, nicht aber das Zusammenspiel der Benutzerschnittstellenfragmente wie es im BSF-AIN formalisiert ist.

Eine Aufgabe der Komposition ist es, unter Berücksichtigung der in Abschnitt 8.4.1 beschriebenen Problemstellung das Zusammenspiel der Benutzerschnittstellenfragmente im BSF-AIN zu untersuchen. Insbesondere die modellierte parallele Ausführung von Interaktionen kann dazu führen, dass aufgrund der begrenzten Möglichkeiten zur Darstellung Benutzerschnittstellenfragmente nicht gleichzeitig angezeigt werden können. Abhängig vom Nutzungsprofil ist es sogar zu erwarten, dass im BSF-AIN vermehrt Parallelität modelliert wurde, wenn Interaktionen möglichst flexibel angeboten werden sollen. Das Auflösen paralleler Interaktionen für die Generierung von Benutzerschnittstellen ist daher eine wichtige Aufgabe der Komposition.

Um diese Aufgabenstellung zu bewältigen, wird im Folgenden ein Algorithmus skizziert, mit dessen Hilfe ermittelt werden kann, ob die Benutzerschnittstellenfragmente und ihr Zusammenspiel, wie es im BSF-AIN modelliert wurde, auf dem gegebenen Layoutpattern ausgeführt werden können. Der Algorithmus stellt weiterhin sicher, dass falls dies nicht möglich ist, das BSF-AIN entsprechend modifiziert wird.

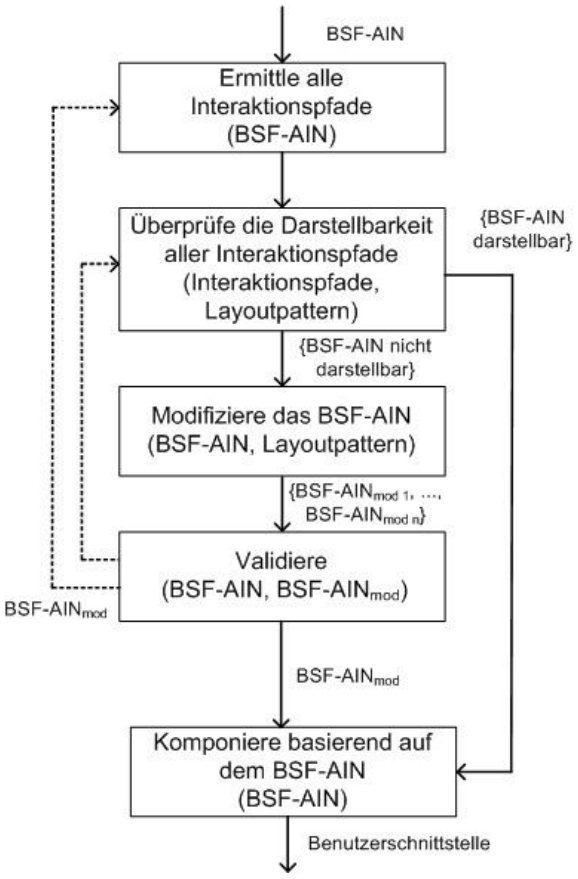


Abbildung 49: Skizze des Kompositionsalgorithmus

Abbildung 49 skizziert solch einen Algorithmus. Dabei beschreibt jedes Rechteck einen Arbeitsschritt, der durch eine Direktive (Text im Rechteck) und die notwendigen Parameter für die Ausführung (Text in Klammern) definiert ist. Die gerichteten Kanten beschreiben den Ablauf der Direktiven, und die Kantenbeschriftungen definieren, welche Informationen weitergeleitet werden bzw. wann ein bestimmter Pfad zu wählen ist. Die beiden „gepunkteten" Kanten beschreiben Methodenaufrufe im Rahmen der Bearbeitung des Arbeitsschrittes „Validiere". Dem Algorithmus liegt dabei folgendes Vorgehen zu Grunde: Das BSF-AIN wird analysiert („Ermittle alle Interaktionspfade" und „Überprüfe die Darstellbarkeit aller Interaktionspfade"), um festzustellen, ob es auf dem gegebenen Nutzungsprofil darstellbar ist. Wenn es darstellbar ist, wird basierend auf dem BSF-AIN komponiert. Andernfalls wird eine Menge von modifizierten BSF-AINs ($BSF\text{-}AIN_{mod}$ in Abbildung 49) erzeugt. Jedes dieser modifizierten BSF-AINs wird untersucht („Validiere" in Abbildung 49), um festzustellen, ob es die Eigenschaften des ursprünglichen BSF-AIN noch korrekt abbildet. Ist dies nicht der Fall, so wird es verworfen. Alle übrigen $BSF\text{-}AIN_{mod}$ werden mit Hilfe der vorhergegangenen Methoden erneut validiert und überprüft, ob diese auf dem Nutzungsprofil darstellbar sind. Das $BSF\text{-}AIN_{mod}$, das auf dem Nutzungsprofil darstellbar ist und die wenigsten Modifikationen aufweist, wird für die Komposition herangezogen. Im Folgenden wird das Vorgehen in den einzelnen Arbeitschritten erläutert.

Im ersten Schritt **„Ermittle alle Interaktionspfade“** werden alle möglichen Interaktionspfade im BSF-AIN ermittelt (vgl. Anhang D). Ein Interaktionspfad eines BSF-AINs beschreibt dabei eine Möglichkeit für die Durchführung des BSF-AIN in Form einer Sequenz von Interaktionen (beschrieben durch BSF). Die Menge aller möglichen Sequenzen beschreibt vollständig jeden validen (beginnend mit der Startmarkierung) Zustand, den das BSF-AIN annehmen kann. Jedem Interaktionspfad kann nicht nur entnommen werden, welche Benutzerschnittstellenfragmente ausgeführt werden, sondern auch, welche Benutzerschnittstellenfragmente gleichzeitig sichtbar sind. Diese Informationen werden für den nächsten Schritt **„Überprüfe die Darstellbarkeit aller Interaktionspfade“** verwendet, um zu ermitteln, ob alle Interaktionspfade auf dem vorgegebenen Layoutpattern abbildbar sind. Existiert ein Interaktionspfad, der nicht abbildbar ist, bedeutet dies, dass eine Generierung der Benutzerschnittstelle nicht möglich ist. Methoden und Werkzeuge für die Ermittlung und Bewertung von Interaktionspfaden werden in Abschnitt 8.4.3 bereit gestellt.

Kann für das gegebene BSF-AIN keine Benutzerschnittstelle generiert werden, so muss es modifiziert werden. Im Arbeitschritt **„Modifiziere das BSF-AIN“** werden hierzu parallele Abläufe im BSF-AIN sequentiell abgebildet. Das Ergebnis ist eine Menge von modifizierten BSF-AIN_{mod}. Jedes BSF-AIN_{mod} beinhaltet eine mögliche Variation des ursprünglichen BSF-AIN. Bevor die BSF-AIN_{mod} wieder durch „Ermittle alle Interaktionspfade“ und „Überprüfe die Darstellbarkeit aller Interaktionspfade“ untersucht werden, wird durch die Methode **„Validiere“** überprüft, ob durch die Modifikation im BSF-AIN_{mod} alle anderen Eigenschaften des ursprünglichen BSF-AINs geblieben sind.

Das am besten geeignete BSF-AIN_{mod} wird für die Komposition herangezogen, wobei der Variante der Vorzug gegeben wird, die mit den geringsten Modifikationen erstellt wurde. Für den hier präsentierten Algorithmus ist anzumerken, dass es immer eine Lösung geben wird. Die sequentielle Ausführung aller BSF ist immer realisierbar, da jedes einzelne BSF auf dem Layoutpattern dargestellt werden kann (vgl. Bewertungsfunktion in Abschnitt 8.2.3).

8.4.3 Analyse und Bewertung von Interaktionspfaden im BSF-AIN

Der im Abschnitt 8.4.2 vorgestellte Algorithmus zur Komposition sieht im ersten Schritt eine Analyse aller Interaktionspfade des BSF-AIN vor. Ziel ist es zu ermitteln, ob das BSF-AIN über einen Interaktionspfad verfügt, der sich nicht auf das Layoutpattern abbilden lässt.

Für die Ermittlung der Interaktionspfade wird die Syntax des dem BSF-AIN zugrundeliegenden AINs herangezogen. Im Folgenden wird deshalb basierend auf den Elementen eines AINs (Stellen, Transitionen, Kanten und Markierungen im Netz) argumentiert, wobei gilt:

- Stellen repräsentieren Präsentationsfragmente.
- Transitionen repräsentieren BSF.
- Kanten repräsentieren die Möglichkeit zur Ausführung eines BSF, wie es im AIN definiert wurde.

- Markierungen in den Stellen repräsentieren sichtbare Präsentationsfragmente (und damit auch eine Belegung auf dem Layoutpattern).

Analysiert werden die Interaktionspfade mit Hilfe eines Interaktionsbaumes, der wie folgt definiert ist:

Ein Interaktionsbaum ist ein Baum für den gilt:

> *(1) Die Wurzel des Interaktionsbaumes ist die leere Menge und besitzt nur eine Kante, die durch die Starttransition definiert ist.*
>
> *(2) Ein Knoten enthält eine Menge bestehend aus Stellen und Transitionen des BSF-AINs.*
>
> *(3) Jede Kante ist mit dem Namen einer Transition (Interaktion) beschriftet. (Welche Interaktionen möglich sind, wird durch die Menge der Stellen und Transitionen im Vater-Knoten definiert).*

Basierend auf dieser Definition wird ein Interaktionsbaum wie folgt aufgebaut: durch die Starttransition werden Stellen im AIN markiert, wobei die Menge dieser Stellen den ersten Knoten definiert. Für alle Transitionen, die in der gegebenen Markierung schalten können, werden Kanten generiert, wobei jede Kante mit der verantwortlichen Transition gekennzeichnet wird. Der „Wert“ der neuen Knoten ergibt sich aus der Menge des Vater-Knotens und der Transition, die gestartet wurde.

Bei der Erstellung des Interaktionsbaums ist dabei zu beachten, dass jede Transition in zwei Schritten schaltet. Beim ersten Schalten wird die Transition ausgeführt. Dies bedeutet für die Betrachtung des Layoutpatterns, dass sie Bereiche einnehmen kann. Dieser Zustand wird innerhalb eines Knoten abgebildet, indem die Transition mit in den Knoten aufgenommen wird, die Eingangsstellen (auch Nebenstellen) aber aus dem Knoten entfernt werden. Das zweite Schalten einer Transition beschreibt das Beenden des BSF. Das Ergebnis des beendenden Schaltens wird aus dem BSF-AIN ermittelt, indem alle Ausgangsstellen (auch Nebenstellen) der Transition, wie sie für AINs definiert sind, markiert werden.

Abbildung 50 zeigt einen Ausschnitt eines Interaktionsbaums am Beispiel des BSF-AINs, das in Abbildung 46 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“) präsentiert wurde. Dargestellt wird nur ein Pfad im Interaktionsbaum mit seinen möglichen Alternativen, wobei die Transition „Mahlzeit planen“ als Starttransition fungiert. Der verfolgte Pfad ist geschwärzt dargestellt, während die Alternativen, die im weiteren Verlauf nicht weiter betrachtet werden, ausgegraut wurden. Zur besseren Visualisierung sind Knoten mit geschweiften und Transitionen mit eckigen Klammern gekennzeichnet.

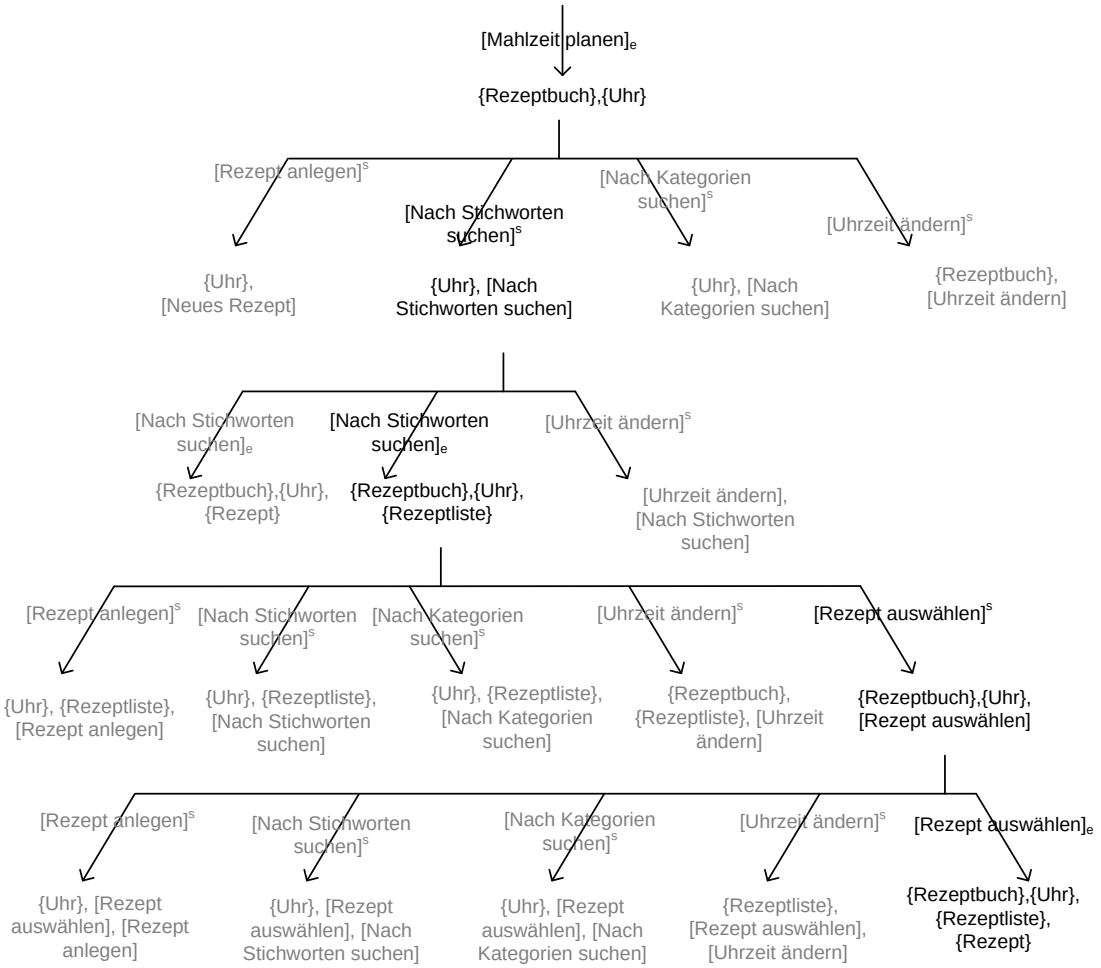


Abbildung 50: Ausschnitt eines Interaktionspfades im Interaktionsbaum eines BSF-AIN

Anhand des Abbildung 50 präsentierten Interaktionsbaums lassen sich einige Besonderheiten für die Generierung veranschaulichen. In einem ersten Schritt wird die Starttransition „Mahlzeit planen“ ausgeführt. Das „$_e$“ in der Transition „Mahlzeit planen“ deutet an, dass damit das Beenden der Transition modelliert wird. Betrachtet man das BSF-AIN aus Abbildung 46 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“), so ist nach den Regeln des AINs ersichtlich, dass nach dem Schalten der Transition die Stellen „Rezeptbuch“ und „Uhr“ markiert sind. Aus dem BSF-AIN kann nun abgelesen werden, dass mit dieser Belegung die vier Transitionen „Rezept anlegen“, „Nach Stichworten suchen“, „Nach Kategorien suchen“ und „Uhrzeit ändern“ schalten können. Entsprechend sind die vier nachfolgenden Kanten zu definieren.

Betrachtet wird davon in Abbildung 50 nur der Knoten, der durch Ausführung der Transition „Nach Stichwort suchen“ generiert wird. Wie man dem Interaktionsbaum entnehmen kann, enthält der Knoten nur die Stelle „Uhr“ und die Transition „Nach Stichworten suchen“. Die Stelle Rezeptbuch ist nicht mehr verfügbar, da die Transition „Nach Stichworten suchen“ diese Stelle als Nebenstelle hat. Diese Situation

entspricht dem im BSF-AIN modellierten Verhalten. Solange der Nutzer sich in einer Alternative befindet, sind keine anderen verfügbar.

Zur weiteren Ermittlung des Pfades werden von diesem Knoten aus wieder die Kanten (wie oben beschrieben) durch die möglichen Transitionen im BSF-AIN ermittelt (im Beispiel ist dies nur die Transition „Uhrzeit ändern“). Zusätzlich kann natürlich die Transition „Nach Stichworten suchen“ beendend schalten, welches ebenfalls durch eine Kante modelliert werden muss. Zu beachten ist, dass vom Knoten „{Uhr}, [Nach Stichworten suchen]“ zwei unterschiedliche Kanten mit derselben Beschriftung „Nach Stichworten suchen“ zu finden sind. Dies ist möglich, da es sich bei dieser Transition um eine „benutzerinitiierte Systemtransition“ (vgl. Abschnitt 5.3.3) handelt, bei der nicht vorhergesagt werden kann, wie das Ergebnis der Transition ausfällt. Für die Analyse der Interaktionspfade stellt dies keine Problemstellung dar, da die Inhalte der Knoten untersucht werden, die in beiden Fällen klar zu unterscheiden sind.

Analog zu dem hier beschriebenen Verfahren kann ein vollständiger Interaktionsbaum automatisiert aufgebaut werden. Um Kreise im Interaktionsverlauf zu vermeiden, muss bevor ein Knoten auf seine Kanten hin untersucht wird, überprüft werden, ob solch ein Knoten (definiert durch die Menge an Stellen und Transitionen) schon im Interaktionsbaum existiert. Ist dies der Fall, braucht der Knoten nicht weiter betrachtet werden, da ein Zustand im Interaktionsablauf erreicht wurde, der schon bekannt ist. Schließlich werden die Blätter des Interaktionsbaumes durch die Endtransitionen definiert.

Betrachtet man die Knoten des Interaktionsbaums, so beschreiben diese alle möglichen Markierungen (inklusive der gestarteten Transaktionen), die im Interaktionsablauf auftreten können. Die Beantwortung der Fragestellung, ob eine Markierung existiert, die sich nicht auf das Layoutpattern abbilden lässt, ist durch die Betrachtung aller Knoten im Interaktionsbaum möglich.

> *Existiert im Interaktionsbaum eines BSF-AINs ein Knoten, und damit eine Menge an Stellen und Transitionen, die sich nicht auf das Layoutpattern abbilden lässt, so ist das BSF-AIN auf dem gegebenen Layoutpattern nicht zu realisieren.*

Jeder Knoten definiert eine Menge aus Stellen und Transitionen, denen im BSF-AIN Präsentationsfragmente und BSF eindeutig zugeordnet sind. Letztere geben eine eindeutige Belegung auf dem Layoutpattern vor. Finden sich Überschneidungen (ein oder mehrere BSF bzw. PF belegen den gleichen Bereich auf dem Layoutpattern), so ist das BSF-AIN nicht auf dem Layoutpattern abbildbar und folglich eine Generierung der Benutzerschnittstelle nicht möglich.

Eine Besonderheit bilden dabei Ausgangsstellen von Transitionen. Da in BSF-AIN Transitionen unabhängig von ihren Ausgangsstellen schalten dürfen, müssen diese Stellen aus der Betrachtung der jeweiligen Knoten im Interaktionsbaum herausgezogen werden. Haben eine Transition und eine oder mehrere ihrer Ausgangstellen sich überschneidende Bereiche im Layoutpattern, so dürfen die Bereiche dieser Stellen vernachlässigt werden, da sie nach dem Schalten sowieso überschrieben werden.

Im folgenden Abschnitt wird eine Methodik entwickelt, die aufzeigt, wie automatisiert Modifikationen am BSF-AIN durchgeführt und validiert werden können.

8.4.4 Sequenzialisierung von parallelen Abläufen im BSF-AIN

Wurde festgestellt, dass das BSF-AIN nicht auf dem Layoutpattern realisierbar ist, so muss versucht werden, das BSF-AIN entsprechend zu modifizieren. Da jedes Benutzerschnittstellenfragment einzeln betrachtet auf dem Layoutpattern abgebildet werden kann (vgl. Algorithmus in Abschnitt 8.2.3), muss die Ursache für die „Nicht-Darstellbarkeit“ in der parallelen Darstellung von Benutzerschnittstellenfragmenten im BSF-AIN zu finden sein. Ziel ist es, solche parallelen Abläufe zu identifizieren und aufzulösen.

Die Stellen, an denen parallele Abläufe im BSF-AIN beginnen, sind einfach zu identifizieren. Verantwortlich sind Transitionen, die mehr als eine Ausgangsstelle besitzen. Nebenstellen zählen dabei ebenfalls als zusätzliche Ausgangsstellen. Das Schalten solch einer Transition markiert zwei Stellen im BSF-AIN, deren Transitionen dann ebenfalls „parallel“ schalten können.

Auch wenn sich der Anfang von parallelen Abläufen einfach identifizieren lässt, ist das Auflösen nicht trivial, da das BSF-AIN verändert werden muss und dadurch Inkonsistenzen auftreten können. Diese Inkonsistenzen beschränken sich nicht nur auf die syntaktische Ebene des zugrundeliegenden AINs, sondern können auch die Semantik des BSF-AINs verändern.

Eine allgemeine Lösung dieses Problems ist komplex. Durch das BSF-AIN wird ein Graph definiert, der durch „Events“ (im Fall des BSF-AINs sind das Interaktionen des Nutzers) gesteuert wird. So definieren markierte Stellen im BSF-AIN zwar, welche Transitionen schalten können, es ist aber nicht vorsehbar, welche Transition als nächstes schalten wird. Durch das Layoutpattern und die Belegung der Bereiche des Layoutpatterns durch Stellen und Transitionen werden weiterhin „Constraints“ definiert, welche bei der Ausführung des BSF-AIN erfüllt bleiben müssen.

Diese Problemstellung kann als ein sogenanntes „dynamic CSP“ (dynamic Constraint Satisfaction Problem) (vgl. [VS94]) beschrieben werden. Durch das BSF-AIN und die möglichen Markierungen im Netz wird ein dynamisches System (im BSF-AIN sind die Schaltregeln der Transitionen statische und die Markierungen der Stellen im Netz dynamische Regeln) definiert. Das Layoutpattern und die Bereiche, die Stellen und Transitionen belegen, definieren schließlich die Constraints, die eingehalten werden müssen. Ziel ist es, das BSF-AIN so zu modifizieren, das die Constraints erfüllt werden. Die am BSF-AIN durchgeführten Modifikationen, sollen dabei das durch das BSF-AIN formulierte Regelsystem weitgehend erhalten.

Als existierende Lösungsansätze für das „dynamic CSP“ gelten (vgl. [VS94]):

- „Incremental Search Methods“ (vgl. [HP91])
  Methoden, die versuchen, erzielte Teillösungen zu bewahren und durch Modifikation eine vollständige Lösung zu ermitteln. Ein Algorithmus lässt sich wie folgt skizzieren: Die Konflikte werden in Form einer Liste notiert. Nachdem eine Lösung für den ersten Konflikt gefunden wurde, wird versucht für den zweiten Konflikt eine Lösung zu finden, ohne dass der erste Konflikt erneut auftritt. Dies wird solange fortgeführt bis für den akutell zu bearbeitenden Konflikt keine Lösung mehr angegeben werden kann. Über ein Backtracking werden dann andere Lösungen für die vorhergehenden Konflikte gesucht.
- „Local Repair Methods“ (vgl. [MJPL92])
  Methoden, die eine vorhergehende Lösung (bei gegebenen Constraints exisitieren keine Konflikte) als Ausgangslage nehmen und versuchen, durch Modifikation dieser Lösung existierende Konflikte (die sich durch geänderte Constraints ergeben) zu beheben.
- „Constraint Recording Methods“ (vgl. [SV93])
  Methoden, die existierende Modifikationen, die zu einer Lösung geführt haben, aufzeichnen und diese beispielsweise im Rahmen von Backtracking-Algorithmen anwenden.

Eine direkte Anwendung der genannten Lösungsansätze auf die gegebene Problemstellung erfordert eine eingehende Analyse des BSF-AINs, eine Bewertung der dort modellierten Regeln und eine Untersuchung möglicher Modifikationen des BSF-AINs. Dabei ist sogar fraglich, ob eine Bewertung der Modifikationen automatisiert erfolgen kann, oder menschliche Experten benötigt werden.

Im Rahmen dieser Arbeit wird daher nur ein erster pragmatischer Lösungsansatz vorgestellt, der auf der Grundidee der „Incremental Search Method“ beruht und der Platz für zukünftige Optimierungen lässt. Zum Auflösen paralleler Abläufe werden fünf Methoden erarbeitet, die auf typischen Modellierungskonstrukten beruhen, wie sie für die Beschreibung im BSF-AIN eingesetzt werden. Diese Methoden werden nun auf solche Konstrukte im BSF-AIN angewendet und anschließend überprüft, ob das BSF-AIN noch den Regeln des ursprünglichen Modells entspricht. Ist dies nicht der Fall, so wird die Änderung rückgängig gemacht und an einer anderen Stelle erneut probiert. Es wird also ein „Brute-Force“-Ansatz verfolgt, der mögliche Modifikationen des Netzes vornimmt und erst im Nachhinein bewertet, ob dadurch das Ziel erreicht wurde. In einem ersten Schritt werden die fünf Methoden zur Modifikation des BSF-AINs beschrieben und anschließend dargestellt, wie die Zielerreichung der Modifikation überprüft werden kann.

1) Überschreiben von Interaktionsabläufen
   Eine offensichtliche Möglichkeit zur Sequenzialisierung von parallelen Abläufen im BSF-AIN liegt darin begründet, Bereiche des Layoutpatterns nur durch die aktuelle Interaktion belegen zu lassen. Das bedeutet aber auch, dass Bereiche, die durch vorhergegangene Interaktionen belegt wurden, durch aktuelle überschrieben werden.
   Realisiert wird diese Form der Sequenzialisierung durch eine Erweiterung des Schaltverhaltens von Transitionen. Kann eine Transition nicht schalten, weil für sie notwendige Bereiche des Layoutpatterns durch Stellen oder vorher gestartete Transitionen belegt sind, so werden diese beendet und

dadurch die entsprechenden Bereiche freigegeben. Teilweise lässt sich diese Erweiterung des Schaltverhaltens auf das BSF-AIN übertragen, indem die Markierungen aus bestimmten Stellen (falls notwendig) abgezogen werden. Da die graphische Repräsentation des BSF-AINs keine Kanten zwischen Transitionen vorsieht, wird das erweiterte Schaltverhalten separat notiert.
Ermittelt werden kann das erweiterte Schaltverhalten mit Hilfe des Interaktionsbaumes des BSF-AINs. Wird ein Knoten im Interaktionsbaum ermittelt, der sich nicht auf das Layoutpattern abbilden lässt, so wird ein erweitertes Schaltverhalten für die Transition definiert, die diesen Knoten erzeugt hat. Im Rahmen des erweiterten Schaltverhaltens werden Stellen und Transitionen entfernt, welche Bereiche des Layoutpatterns belegen, die von der Transition oder deren Ausgangsstellen benötigt werden. Dieses erweiterte Schaltverhalten ist nun immer anzuwenden, wenn die Transition schaltet. Dabei entstehen keine zusätzlichen Bedingungen für das Schalten der Transition. Das heißt, dass diese Transition auch schalten kann, wenn die zu entfernenden Stellen oder Transitionen gar nicht verfügbar sind. Das erweiterte Schaltverhalten kann für das Starten und das Beenden der Transition definiert werden.

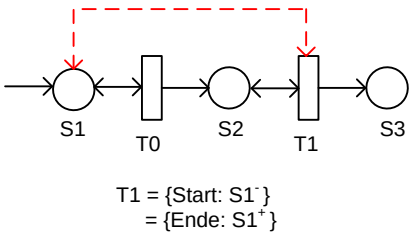


Abbildung 51: Überschreiben von Interaktionsabläufen

Abbildung 51 visualisiert die Methode anhand eines Beispiels. Belegen die Transition T1 und die Stelle S1 überschneidende Bereiche im Layoutpattern, so kann das Schaltverhalten von T1 wie dargestellt verändert werden. $S1^-$ bedeutet dabei, dass die Bereiche im Layoutpattern von S1 freigegeben werden und nach dem Beenden von T1 wieder hergestellt ($S1^+$ in Abbildung 51) werden. Die rot dargestellte Kante im AIN in Abbildung 51 visualisiert das veränderte Schaltverhalten. Durch diese Kante wird keine zusätzliche Bedingung definiert, das heißt, T1 kann schalten, wenn S2 markiert ist, unabhängig davon ob S1 markiert ist oder nicht.

2) Auflösen von Nebenstellen

Wie man dem Beispiel in Abbildung 46 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“) entnehmen kann, werden Nebenstellen verwendet, um darzustellen, dass Benutzer bestimmte Interaktionen immer wieder durchführen, und gegebenenfalls auch Alternativen ausprobieren können. So ist das „Rezeptbuch“ in dem Beispiel gleichzeitig Nebenstelle für die Transitionen „Rezept anlegen“, „Nach Stichworten suchen“ und „Nach Kategorien suchen“. Daher wird als eine Möglichkeit zum Sequenzialisieren vorgeschlagen, eine Nebenstelle in eine „reine“ Eingangsstelle umzuwandeln. Dies bedeutet für den Benutzer eine Einschränkung in der Flexibi-

lität der Interaktionsdurchführung, da er, sobald er einen Pfad eingeschlagen hat, nicht parallel weitere Alternativen wahrnehmen kann.

3) Auflösen von Nebenstellen durch Sequenzen

Abbildung 52 visualisiert. wie eine Nebenstelle (für zwei Transitionen) sequenzialisiert werden kann. Dabei wird die Nebenstelle (gekennzeichnet durch das „s“) dupliziert und zusätzlich eine neue „primitive Transition“ eingeführt. „Primitive Transitionen“ realisieren nur ein Interaktionsobjekt (bspw. Button) im Präsentationsfragment.

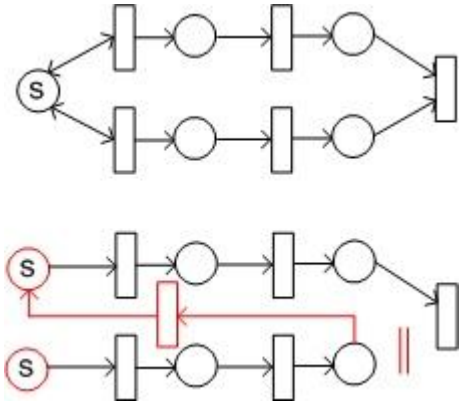


Abbildung 52: Auflösen von Nebenstellen durch Sequenzen

Es lässt sich leicht ein Algorithmus skizzieren, der diese Modifikation durchführt, wenn die Nebenstelle und die Transition, welche die beiden parallelen Abläufe synchronisiert, gegeben sind. Voraussetzung für diese Methode ist, dass eine Nebenstelle existiert, die Nebenstelle von mindestens zwei Transitionen ist. Weiterhin müssen Pfade von den Transitionen dieser Nebenstelle zu den Eingangsstellen der „synchronisierenden Transition“ identifiziert werden können.

Im Gegensatz zur ersten Variante, in welcher die parallele Durchführung von unabhängigen Alternativen eingeschränkt wird, handelt es sich bei dieser Variante nicht mehr um Alternativen. Damit die letzte Transition schalten kann, müssen beide Pfade durchlaufen werden, so dass ein einfaches Auflösen der Nebenstellen zur Folge hätte, dass bestimmte Transitionen gar nicht mehr erreichbar sind.

4) Auflösen durch Alternative

Transitionen mit zwei Ausgangstellen können durch die Einführung einer zusätzlichen Stelle mit zwei „primitiven Transitionen“, wie in Abbildung 53 dargestellt, erweitert werden.

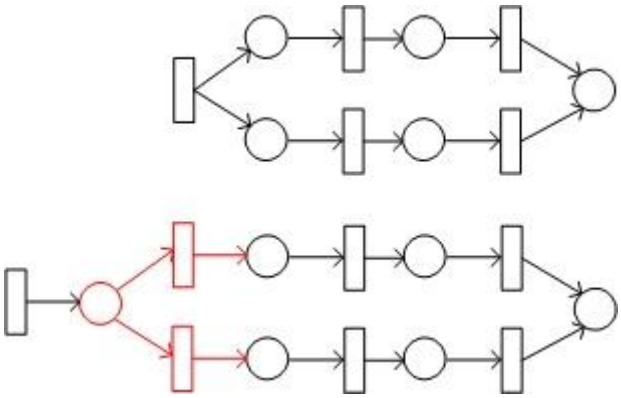


Abbildung 53: Auflösen von parallelen Abläufen durch Alternativen

Dadurch wird analog zum Auflösen der Nebenstellen davon ausgegangen, dass der Modellierer des BSF-AINs Flexibilität in der Interaktionsdurchführung erreichen wollte, nicht aber, dass wirklich beide Zweige im Interaktionsablauf durchlaufen werden müssen.

5) Auflösen durch Sequenzialisieren

Analog zur dritten Methode (Auflösen von Nebenstellen durch Sequenzen) müssen auch Methoden für Transitionen mit zwei oder mehr Ausgangsstellen bereitgestellt werden, welche es erlauben, zwei parallele Pfade in eine Sequenz zu überführen.

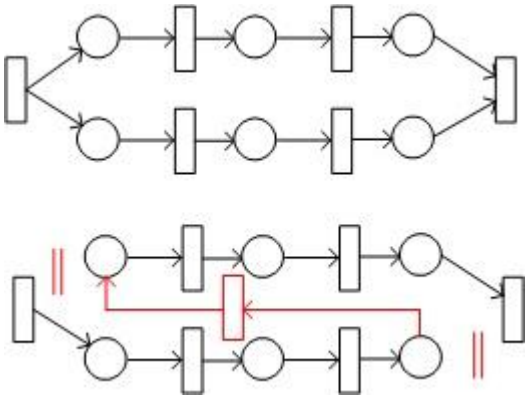


Abbildung 54: Auflösen von parallelen Abläufen durch Sequenzialisierung

Abbildung 54 beschreibt solch eine Möglichkeit, dabei werden zwei Kanten wie in Abbildung 54 dargestellt entfernt und eine zusätzliche „primitive Transition“ eingeführt. Wenn beide Transitionen gegeben sind, lässt sich eine Automatisierung des Verfahrens beschreiben.

Es ist leicht darzustellen, dass die fünf hier vorgestellten Methoden nur für bestimmte Modelle anwendbar sind. Versucht man beispielsweise Methode 2 „einfaches Auflösen von Nebenstellen“ auf das in Abbildung 52 dargestellte Netz anzuwenden, so ist offensichtlich, dass die letzte Transition niemals schalten kann.

Gleiches gilt auch für die Methode 4 „Auflösen durch Alternativen“, wenn sie auf das Netz in Abbildung 54 angewendet wird. Neben der Fragestellung, welche Methoden auf welche Netze angewendet werden können, ist auch die Anwendung der Methoden nicht in allen Fällen eindeutig zu klären. So zeigt das Netz in Abbildung 54, dass die Methode „Auflösen von parallelen Abläufen durch Sequenzialisierung“ die richtige Wahl der aufzulösenden Kanten voraus setzt. In dem dargestellten Beispiel ist dies zwar einfach zu ermitteln, kann aber in komplexen Netzen höchst schwierig sein. Insbesondere Alternativen in der Interaktionsdurchführung können dazu führen, dass nicht mehr klar erkennbar ist, welche Pfade parallel ausgeführt werden sollen.

Möglich sind solche „unklaren“ Konstrukte, da auch hier der Modellierer des Netzes nicht notwendigerweise auf Stringenz und Effizienz, sondern vielmehr auf Flexibilität der Interaktionsdurchführung Wert gelegt hat. Auch wenn das Auffinden der „synchronisierenden“ Transitionen (mit zwei oder mehr Eingangsstellen) und der „parallel startenden“ Transitionen (mit zwei oder mehr Ausgangsstellen) im Netz einfach ist, kann sich bereits der Versuch einer Zuordnung als schwierig erweisen.

Es kann versucht werden, Algorithmen zu definieren, die basierend auf Pfadanalysen im Interaktionsbaum die oben beschriebenen Fragestellungen beantworten können. Da aber bereits jetzt vorhersehbar ist, dass die fünf angegebenen Methoden nur einen kleinen Ausschnitt möglicher Varianten beschreiben, wurde entschieden, einen „Brute-Force“-Ansatz (auch Exhaustionsmethode genannt) zu definieren, bei dem alle möglichen Varianten erprobt werden, und für jede Variante überprüft wird, ob diese zulässig ist oder nicht. Vorteil dieses Verfahrens ist, dass beliebig weitere Methoden aufgenommen werden können, wobei der Kern des „Brute-Force“-Ansatzes, nämlich die Überprüfung, ob das Ergebnis zuverlässig ist oder nicht, beibehalten wird.

Jede Modifikation des BSF-AINs verändert das Netz und muss überprüft werden. Ein Netz gilt dabei als zulässig, wenn die folgenden vier Bedingungen erfüllt sind:

- Alle Transitionen müssen weiterhin erreichbar sein. Das bedeutet, dass im Interaktionsbaum das Schalten jeder Transition möglich ist.
- Invarianten von Bedingungen im BSF-AIN (definiert durch Transitionen mit mehr als einer Eingangsstelle) müssen eingehalten werden. Eine Transition mit mehr als einer Eingangsstelle definiert eine Bedingung, welche besagt, dass beide Stellen markiert sein müssen, damit diese Transition schalten kann. Das bedeutet aber, dass andere Transitionen „vorher“ schalten müssen. Für die Interaktionsdurchführung kann man daher von einer Invariante sprechen, die für alle Interaktionspfade eingehalten werden muss.
- Es darf keine (zusätzlichen) Pfade geben, die nicht zur Endtransition führen.
- Es darf keine Kreise im Interaktionsablauf geben, die nicht mehr verlassen werden können und nicht zur Endtransition führen.

Die ersten beiden Bedingungen können direkt anhand des Interaktionsbaumes abgelesen werden. Dabei muss nur überprüft werden, ob alle Transitionen im Interaktionsbaum vorkommen, und ob auf allen Pfaden die Invarianten erfüllt sind.

Pfade und Kreise im Interaktionsablauf, die nicht zur Endtransition führen, werden identifiziert, indem jeder Pfad im Interaktionsbaum von der Endtransition bis zur Wurzel rückwärts durchlaufen, und dabei jeder Knoten auf diesem Pfad markiert wird. Existieren noch Knoten, die nicht markiert wurden, so existieren Pfade im BSF-AIN, die von der Starttransition erreicht werden können, aber nicht zum Beenden des BSF-AIN führen.

Mit Hilfe dieses Verfahrens kann eine Methode zur Validierung des BSF-AINs angegeben werden, mit der überprüfen werden kann, ob Modifikationen des BSF-AINs noch dem ursprünglichen Netz folgen. Dieses Verfahren wird jetzt benutzt, um die fünf Methoden zur Auflösung von parallelen Abläufen im BSF-AIN sukzessive auf ein BSF-AIN anzuwenden. Ziel ist es, solange Modifikationen durchzuführen, bis ein BSF-AIN erreicht wurde, das sich auf dem gegeben Layoutpattern realisieren lässt. Ein entsprechender Ansatz wird im Folgenden skizziert. Dabei wird der Begriff „gültig“ für ein BSF-AIN verwendet, um zu beschreiben, dass die Modifikationen im Bezug auf das Ursprungsnetz zulässig sind. Es wird der Begriff „positiv validiert“ verwendet, wenn das BSF-AIN „gültig“ ist und sich auf das Layoutpattern abbilden lässt.

Ausgangslage des Verfahrens ist der Interaktionsbaum, der aus dem BSF-AIN generiert wird. Die Anzahl der Knoten, die nicht auf das Layoutpattern abgebildet werden können, wird als Erfolgsfaktor der Modifikationen herangezogen.

Sei Q die Menge der Knoten im Interaktionsbaum, die nicht auf das Layoutpattern abgebildet werden können und Q# die Anzahl der Elemente in Q:

1. *Schritt 1 (Anwendung Methode 1)*
   Wenn $Q\# \neq 0$
   Wende für jede Teilmenge von Q die Methode 1 an.
   - Existiert eine oder mehrere Teilmengen die positiv validiert wurden, so definieren die Teilmengen mit den wenigsten Elementen das Ergebnis.
   - Konnte kein Ergebnis ermittelt werden, so wird versucht mit Hilfe der Methode 1 ein BSF-AIN zu generieren, dessen Interaktionsbaum über weniger Knoten verfügt, die nicht auf das Layoutpattern abgebildet werden können. Hierzu wird für jede Teilmengen q aus Q ein BSF-AIN generiert und überprüft, ob dieses gültig ist. Für alle q die gültig sind, wird ein BSF-AIN ermittelt und überprüft, wie viele Knoten im Interaktionsbaum des BSF-AINs nicht auf das Layoutpattern abgebildet werden können. Für das BSF-AIN mit den wenigsten Knoten, wird Schritt 2 aufgerufen. Existieren mehr als ein BSF-AIN, das diese Bedingung erfüllt, so wird das BSF-AIN ausgewählt, für dessen Generierung ein q mit möglichst wenigen Elementen benutzt wurde.

2. *Schritt 2 (Anwendung Methode 2)*
   N sei die Menge aller Nebenstellen im BSF-AIN. Wende für jede Teilmenge aus N die Methode 2 an.
   - Existiert eine oder mehrere Teilmengen, die positiv validiert wurden, so definieren die Teilmengen mit den wenigsten Elementen das Ergebnis.
   - Konnte kein Ergebnis ermittelt werden, so wird versucht, mit Hilfe der Methode 2 ein BSF-AIN zu generieren, dessen Interaktionsbaum über weniger Knoten verfügt, die nicht auf das Layoutpattern abgebildet werden können. Hierzu wird für jede Teilmenge n aus N ein BSF-AIN generiert und überprüft, ob dieses gültig ist. Für alle n die gültig sind, wird ein BSF-AIN ermittelt und überprüft, wie viele Knoten im Interaktionsbaum des BSF-AINs nicht auf das Layoutpattern abgebildet werden können. Für das BSF-AIN mit den wenigsten Knoten, wird Schritt 3 aufgerufen. Existiert mehr als ein BSF-AIN, das diese Bedingung erfüllt, so wird das BSF-AIN ausgewählt, für dessen Generierung ein n mit möglichst wenigen Elementen benutzt wurde.

3. *Schritt 3 (Anwendung Methode 4)*
   P sei die Menger aller Transitionen mit zwei oder mehr Ausgangsstellen im BSF-AIN.
   - Wende für jede Teilmenge von P die Methode 4 an. Existiert eine oder mehrere Teilmengen die positiv validiert wurden, so definieren die Teilmengen mit den wenigsten Elementen das Ergebnis.
   - Konnte kein Ergebnis ermittelt werden, so wird versucht, mit Hilfe der Methode 4 ein BSF-AIN zu generieren, dessen Interaktionsbaum über weniger Knoten verfügt, die nicht auf das Layoutpattern abgebildet werden können. Hierzu wird für jede Teilmengen p aus P ein BSF-AIN generiert und überprüft, ob dieses gültig ist. Für alle p die gültig sind, wird ein BSF-AIN ermittelt und überprüft, wie viele Knoten im Interaktionsbaum des BSF-AINs nicht auf das Layoutpattern abgebildet werden können. Für das BSF-AIN mit den wenigsten Knoten wird Schritt 4 aufgerufen. Existieren mehr als ein BSF-AIN, das diese Bedingung erfüllt, so wird das BSF-AIN ausgewählt, für dessen Generierung ein p mit möglichst wenigen Elementen benutzt wurde.

4. *Schritt 4 (Anwendung Methode 5)*
   P sei die Menger aller Transitionen mit zwei oder mehr Ausgangsstellen und S die Menge aller Transitionen mit zwei oder mehr Eingangsstellen im BSF-AIN.
   - Wende die Methode 5 für alle Tupel (s, p) an, wobei s ein Element von S und p ein Element von P ist. Die Lösungsmenge wird durch alle Tupel definiert, für die Methode 5 ein positiv validiertes Netz liefert.
   - Konnte kein Ergebnis ermittelt werden, so wird versucht, mit Hilfe der Methode 5 ein BSF-AIN zu generieren, dessen Interaktionsbaum über weniger Knoten verfügt, die nicht auf das Layoutpattern abgebildet werden können. Hierzu wird für jede Teilmengen t aus der Menge aller Tupel (s, p) ein BSF-AIN generiert und überprüft, ob dieses gültig ist. Für alle t die gültig sind, wird ein BSF-AIN ermittelt und überprüft, wie viele Knoten im Interaktionsbaum des BSF-AINs nicht auf das Layoutpattern abgebildet werden können. Für das BSF-AIN mit den wenigsten Knoten,

wird Schritt 5 aufgerufen. Existiert mehr als ein BSF-AIN, das diese Bedingung erfüllt, so wird das BSF-AIN ausgewählt, für dessen Generierung ein t mit möglichst wenigen Elementen benutzt wurde.

5. *Schritt 5 (Anwendung Methode 3)*
   N sei die Menge aller Nebenstellen und S die Menge aller Transitionen mit zwei oder mehr Eingangstellen im BSF-AIN.
   - Wende die Methode 3 für alle Tupel (n, s) an, wobei n ein Element von N und s ein Element von S ist. Die Lösungsmenge wird durch alle Tupel definiert, für die Methode 3 ein positiv validiertes Netz liefert.
   - Konnte kein Ergebnis ermittelt werden, so wird versucht, mit Hilfe der Methode 3 ein BSF-AIN zu generieren, dessen Interaktionsbaum über weniger Knoten verfügt, die nicht auf das Layoutpattern abgebildet werden können. Hierzu wird für jede Teilmengen t aus der Menge aller Tupel (n, s) ein BSF-AIN generiert und überprüft, ob dieses gültig ist. Für alle t die gültig sind, wird ein BSF-AIN ermittelt und überprüft, wie viele Knoten im Interaktionsbaum des BSF-AINs nicht auf das Layoutpattern abgebildet werden können. Für das BSF-AIN mit den wenigsten Knoten wird Schritt 1 aufgerufen. Existieren mehr als ein BSF-AIN, das diese Bedingung erfüllt, so wird das BSF-AIN ausgewählt, für dessen Generierung ein t mit möglichst wenigen Elementen benutzt wurde.

Der Ansatz, der in diesem Verfahren verfolgt wird, basiert auf einer Priorisierung der Methoden. Methode 1 verändert zwar das Schaltverhalten, sie greift aber nur da an, wo Konflikte identifiziert wurden und versucht nur für die Durchführung einer Transition in das Netz einzugreifen. Im nächsten Schritt wird versucht, die parallele Durchführung von Alternativen einzuschränken, indem Methode 2 und Methode 4 angewendet werden. Erst wenn dies nicht zu dem gewünschten Ergebnis führt, wird versucht, parallele Abläufe in eine Sequenz zu überführen. Ziel ist es dabei, die Anzahl der Modifikationen möglichst gering zu halten. Insbesondere sollen die beiden Methoden 3 und 5 zur Sequenzialisierung paralleler Abläufe möglichst wenig eingesetzt werden.

Als Kriterium für den erfolgreichen Einsatz einer Methode wird die Anzahl der Knoten im Interaktionsbaum herangezogen, die nicht auf das Layoutpattern abgebildet werden können. Auch wenn eine Methode nicht alle Konflikte auflösen kann, wird doch ein Zwischenergebnis beibehalten, wenn die Anzahl der Konflikte durch diese Methode reduziert wurden. Anschließend wird in weiteren Methoden versucht, die verbliebenen Konflikte zu beheben.

Die Sequenzialisierung des BSF-AIN nach der hier vorgestellten Methode kann nicht eindeutig sein. Auch wenn die „Anzahl der durchgeführten Modifikationen“ ein Kriterium ist, mit dem eine grobe Selektion möglich ist, wird es dennoch unterschiedliche Realisierungsmöglichkeiten geben. Wenn beispielsweise parallele Pfade sequenzialisiert werden, ist syntaktisch nicht zu ermitteln, welcher Pfad idealerweise zuerst zu begehen ist. So wird es unterschiedliche Möglichkeiten geben, die schließlich bewertet werden müssen.

Im Rahmen dieser Arbeit findet die Bewertung von Lösungsalternativen nicht automatisch statt, sondern wird von einem User-Interface Designer durchgeführt. Dieser bewertet allerdings nicht das BSF-AIN, sondern erst die finalen Benutzerschnittstellen. Im Rahmen des Prozessschrittes „Evaluation und Konfiguration“ (vgl. Abschnitt 8.1) wird die Möglichkeit geschaffen, generierte Benutzerschnittstellen direkt miteinander vergleichbar zu machen.

Wendet man dieses Verfahren auf das BSF-AIN aus dem Beispiel in Abbildung 46 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“) an, so ist eines der möglichen Ergebnisse in der Abbildung 55 dargestellt. Das Layoutpattern und auch die Zuordnung von BSF und PF zu den Bereichen des Layoutpatterns ist der Tabelle 11 in Abschnitt 8.4.1 zu entnehmen.

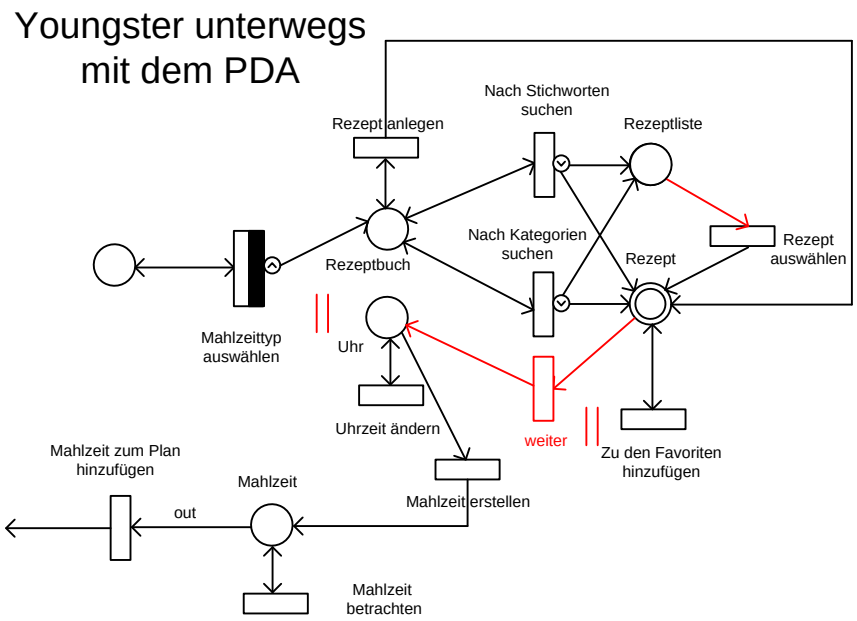


Abbildung 55: BSF-AIN nach der Sequenzialisierung

Die durchgeführten Modifikationen sind rot markiert. Wie man der Abbildung entnehmen kann, wurde Methode 5 (für die Transition „Mahlzeittyp auswählen“ und „Mahlzeit erstellen“) und zusätzlich Methode 2 auf die Nebenstelle der Transition „Rezeptliste“ angewendet. Weiterhin sind die Modifikationen im Schaltverhalten der Transitionen, wie sie mit Methode 1 ermittelt werden, in der folgenden Tabelle 12 dargestellt.

Transition	**Start**	**Ende**
Rezept anlegen	{Rezept}- {Mahlzeit}- [Rezept auswählen]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten]-	
Nach Stichworten suchen	{Rezeptliste}- {Rezept}- {Mahlzeit}- [Rezept auswählen]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten]-	
Nach Kategorien suchen	{Rezeptliste}- {Rezept}- {Mahlzeit}- [Rezept auswählen]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten]-	
Rezept auswählen		
Uhrzeit ändern		
Zu den Favoriten hinzufügen		
Mahlzeit betrachten		

Tabelle 12: Verändertes Schaltverhalten nach Ausführung von Methode 1

Tabelle 12 visualisiert die Änderungen am Schaltverhalten der Transitionen. Dabei wurden die Transitionen in der Reihenfolge, wie sie in der Tabelle angegeben sind, durch die Methode 1 bearbeitet. Die erste Spalte in Tabelle 12 gibt die Transition an, die zweite Spalte („Start“) die Modifikationen, bevor die Transition schaltet, und die letzte Spalte „Ende“ die Modifikationen nach der Ausführung der Transition. „-“ gibt an, dass eine Markierung aus einer Stelle abgezogen wird oder eine gestartete Transition beendet wird. „+“ beschreibt das Wiederherstellen von Markierungen oder gestarteten Transitionen. Stellen wurden durch geschweifte und Transitionen durch eckige Klammern gekennzeichnet.

Eine detaillierte Beschreibung der Durchführung der Sequenzialisierung für das Beispiel „Youngster unterwegs mit dem PDA“, und auch für zwei weitere BSF-AINs findet sich in Kapitel 9. Das generierte BSF-AIN in Abbildung 55 ist auf dem gegebenen Layoutpattern darstellbar und bildet somit die Grundlage für den letzten Schritt des Generierungsprozesses nämlich die Komposition der Benutzerschnittstellenfragmente.

Die Komposition muss sich an den Beziehungen zwischen den Benutzerschnittstellenfragmenten orientieren, wie sie im BSF-AIN definiert sind. Im folgenden Kapitel werden diese Beziehungen eingehender analysiert und mit Hilfe einer Intervallalgebra beschrieben. Jeder Relation der Intervallalgebra werden schließlich Methoden für die Komposition der Benutzerschnittstellenfragmenten zugeordnet.

8.4.5 Komposition der Benutzerschnittstellenfragmente mit Hilfe einer Intervallalgebra

Für die Komposition der Benutzerschnittstellenfragmente wird das BSF-AIN in eine Reihe von Intervallrelationen zerlegt, mit Hilfe derer die Beziehungen der Benutzerschnittstellenfragmente, wie sie im BSF-AIN modelliert sind, abgebildet werden. Jeder Intervallrelation können Methoden zugeordnet werden, welche die Komposition basierend auf der Spezifikation der Benutzerschnittstellenfragmente realisieren. Sowohl die Überführung des BSF-AIN in eine Liste aus Intervallrelationen, als auch die Ausführung der Methoden zur Komposition der Benutzerschnittstellenfragmente können automatisiert erfolgen.

Für die strukturierte Betrachtung des Zusammenspiels von Benutzerschnittstellenfragmenten, wie es im BSF-AIN beschrieben wird, wurde die Intervallalgebra nach Allen [All83] herangezogen. Damit wird in dieser Arbeit ein Ansatz verfolgt, den Limbourg in [LVM*04] für die Beschreibung von Kompositionsmöglichkeiten von Interaktionsobjekten in modellbasierten Benutzerschnittstellenentwicklungsumgebungen vorschlägt. Innerhalb dieser Arbeit erfüllt die Intervallalgebra zwei Aufgaben. Neben der Betrachtung von möglichen Kompositionsbeziehungen im BSF-AIN, wird sie auch als Werkzeug für die Komposition von Benutzerschnittstellenfragmenten verwendet. Eine Intervallalgebra ist durch Zeitintervalle definiert. Ein Zeitintervall I ist ein geordnetes Paar I=(I-,I+) wobei I- und I+ Zeitpunkte auf einer reellen Zeitleiste sind, für die gilt I-<I+. Ein Zeitintervall ist demnach durch seinen Start- und Endpunkt definiert.

Intervall Relation	Symbol	Beziehungen der Endpunkte
I before J	<	$I+ < J-$
I after J	>	$I- > J+$
I meets J	m	$I+ = J-$
I met-by J	mi	$I- = J+$
I overlaps J	o	$I- < J-$, $I+ > J-$, $I+ < J+$
I overlapped-by J	oi	$I- > J-$, $I- < J+$, $I+ > J+$
I during J	d	$I- > J-$, $I+ < J+$
I includes J	di	$I- < J-$, $I+ > J+$
I starts J	s	$I- = J-$, $I+ < J+$
I started-by J	si	$I- = J-$, $I+ > J+$
I finishes J	f	$I- > J-$, $I+ = J+$
I finished-by J	fi	$I- < J-$, $I+ = J+$
I equals J	=	$I- = J-$, $I+ = J+$

Tabelle 13: Intervallalgebra (in Anlehnung an [ZZ99])

Tabelle 13 veranschaulicht die 13 Relationen der Intervallalgebra (interval calculus) nach Allen [All83]. Er führt damit ein Kalkül über Intervalle ein, in dem Wissen über die Zeit durch vergleichende Relationen qualitativ gehandhabt wird. Diese Relationen können zusammengesetzt werden, um Beziehungen zwischen zwei Intervallen darzustellen. Insbesondere interessant an diesem Ansatz ist, dass nicht nur diskrete Zeitpunkte zur Komposition benutzt werden können, sondern auch Intervalle. So definiert beispielsweise die Overlaps-Relation, dass im Laufe des Intervalls I das Intervall J gestartet wird, aber nicht wann genau dies geschehen wird.

Interaktionsabläufe in Benutzerschnittstellenfragmenten können als Folge von zeitdiskreten Ereignissen aufgefasst werden, die somit einen Zeitraum definieren. Der Beginn eines Benutzerschnittstellenfragmentes ist dabei mit der Ausführung des Startdialogfragmentes beschrieben, und das Ende durch Erreichen eines der Enddialogfragmente. Mit dieser Definition ist der Anfang und das Ende eines Benutzerschnittstellenfragmentes ausreichend beschrieben, um die Anforderungen an ein Intervall nach Allen zu erfüllen. Basierend auf den 13 Relationen stehen eine Reihe von Kompositionsmöglichkeiten zur Verfügung, die miteinander verknüpfbar sind, um auch komplexe Kompositionsbeziehungen beschreiben zu können.

Im Folgenden werden nur noch 7 Relationen nach Allen analysiert, da inverse Relationen zu keiner neuen Beziehungsstruktur führen. Bevor die einzelnen Relationen der Intervallalgebra genauer analysiert werden, werden noch die Auswirkungen von Markierungen im BSF-AIN auf die Sichtbarkeit von Benutzerschnittstellenfragmenten erläutert.

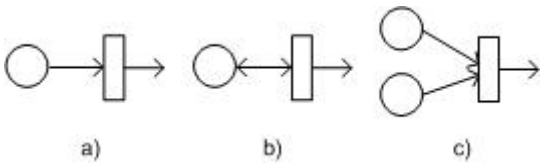


Abbildung 56: Benutzerschnittstellenfragmente im BSF-AIN

Ein Benutzerschnittstellenfragment wird im BSF-AIN als Transition (einschließlich seiner Eingangsstellen) repräsentiert. Dabei gilt, dass die Eingangsstellen die Präsentationsfragmente modellieren, die durch das Startdialogfragment des BSF aktiviert werden. Abbildung 56 visualisiert die Möglichkeiten zur Darstellung eines BSF in einem BSF-AIN.

Teil a) (in Abbildung 56) präsentiert ein BSF im AIN-BSF mit nur einer Eingangsstelle und einer Transition. Die Markierung der Eingangsstelle beschreibt eine Aktivierung des BSF, da die Eingangsstelle ein Präsentationsfragment modelliert, das bereits Teil des BSF ist. Für die Sichtbarkeit von Benutzerschnittstellenfragmenten bedeutet die Markierung, dass das Präsentationsfragment, das durch die Stelle modelliert wird, einen Platz im Layoutpattern belegt. Die Ausführung der Transition wird als Einheit betrachtet und belegt ebenfalls eine Menge von Bereichen des Layoutpatterns. Eine Transition kann auch eine leere Menge an Bereichen im Layoutpattern belegen, falls diese „trivial“ ist, also keine zusätzlichen Präsentationsfragmente, als die durch die Eingangsstelle modellierten, beinhaltet. Nachdem die Ausführung der

Transition abgeschlossen ist, werden alle Bereiche des Layoutpatterns, die durch die Transition und die Stelle belegt wurden, wieder frei gegeben.

Teil b) (in Abbildung 56) zeichnet sich durch eine Nebenstelle aus. Die Betrachtung der Sichtbarkeiten (Belegung der Bereiche des Layoutpatterns) erfolgt analog zu Teil a), mit dem Unterschied, dass nach Ausführung der Transition die Eingangsstelle wieder belegt wird, und damit das BSF die Sichtbarkeit nicht verliert.

Teil c) (in Abbildung 56) veranschaulicht weiterhin, dass die Sichtbarkeit eines BSF differenziert betrachtet werden muss. Verfügt eine Transition über mehrere Eingangsstellen, so ist die Transition nur ausführbar, wenn alle Stellen belebt sind. Für ein BSF bedeutet dies, dass zwar Teile des BSF (nämlich die Präsentationsfragmente, die durch die Eingangsstellen modelliert werden) sichtbar sind, die Transition aber nicht erreichbar ist.

In den folgenden Abschnitten werden die im BSF-AIN modellierten Beziehungen anhand der 7 Relationen der Intervallalgebra diskutiert und beschrieben, wie eine Komposition der Benutzerschnittstellenfragmente nach der angegebenen Relation realisiert werden kann. In jedem Abschnitt wird jeweils eine der Relationen anhand der folgenden Struktur behandelt:

- Kurze Beschreibung der Relation;
- Bedingungen, die für die Komposition berücksichtigt werden müssen;
- Erläuterung der Relation anhand eines Beispiels (incl. Abbildung im BSF-AIN);
- Realisierung der Komposition.

Für die Beschreibung der Komposition wird von BSF_1 und BSF_2 gesprochen, wobei BSF_1 das erste Benutzerschnittstellenfragment (aus der jeweils angegebenen Abbildung eindeutig ersichtlich), und BSF_2 das darauf folgende beschreibt. Die Angabe der Kompositionsregeln erfolgt informal, basiert aber auf der Spezifikationen der BSF wie sie in Abschnitt 7.2.2 definiert wurden. Veranschaulicht werden diese Betrachtungen am Beispiel „BSF-AIN für das Nutzungsprofil Youngster unterwegs mit dem PDA“ (vgl. Abbildung 46).

8.4.5.1 Before & After – Relation

Die Before & After-Relation ist Teil der Intervallalgebra und beschreibt eine Ordnungsreihenfolge von zwei Fragmenten BSF_1 und BSF_2. Dabei sagt diese Relation lediglich etwas über die Abfolge aus, nicht aber, dass das eine Benutzerschnittstellenfragment direkt nach dem anderen ausgeführt werden muss. Folglich können noch andere Benutzerschnittstellenfragmente abgearbeitet werden, und die Relation gilt immer noch als erfüllt.

(BSF_1>BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- BSF_1 wurde gestartet und ist beendet, bevor BSF_2 gestartet werden kann.

- Solange BSF_1 aktiv ist, kann BSF_2 nicht aktiv sein.
- BSF_1 kann Daten bereitstellen, die für BSF_2 relevant sind.

Diese Relation liefert ausschließlich Invarianten für die Komposition, da nicht vorhersagbar ist, welche Interaktionen zwischen BSF1 und BSF2 durchgeführt werden.

8.4.5.2 Meets & Met_by-Relation

Im Gegensatz zur Before&After-Relation, die lediglich eine Aussage über eine Ordnungsbeziehung herstellt, definiert die Meets&Met_by-Relation die direkte Ausführung eines Benutzerschnittstellenfragmentes nach dem anderen. Für ein Benutzerschnittstellenfragment bedeutet dies, dass das letzte Dialogfragment eines Benutzerschnittstellenfragmentes das Startdialogfragment des folgenden Benutzerschnittstellenfragmentes aktiviert.

(BSF_1 m BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- BSF_1 wurde gestartet und ist abgeschlossen, bevor BSF_2 gestartet werden kann.
- Das Dialogfragment, das zum Schließen des Benutzerschnittstellenfragmentes BSF_1 führt, aktiviert das Startdialogfragment von BSF_2.
- Keines der Präsentationsfragmente aus BSF_2 kann aktiv sein, wenn BSF_1 aktiv ist und umgekehrt.

Die Meets-Relation beschreibt die typische sequentielle Ausführung von zwei Benutzerschnittstellenfragmenten. Dabei muss die Ausführung des vorhergehenden Benutzerschnittstellenfragmentes abgeschlossen sein bevor das Folgende starten kann. Die Met_by-Relation wird nicht mehr explizit betrachtet, da für die Komposition von Benutzerschnittstellen keine Unterschiede (zur Meets-Relation) zu identifizieren sind.

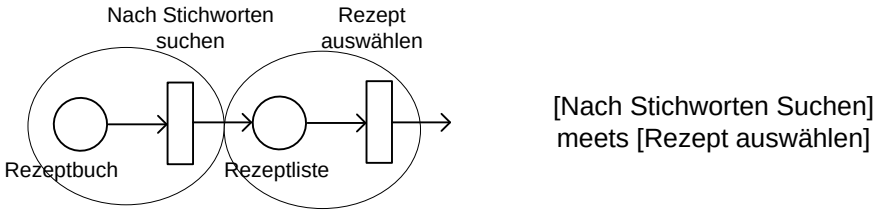


Abbildung 57: Meets-Relation im BSF-AIN

Abbildung 57 zeigt eine Meets-Relation in einem modifizierten Ausschnitt des Beispiels aus Abbildung 46, in welchem alle Nebenstellen ersetzt wurden. Ersichtlich ist, dass in einem ersten Schritt das BSF „Nach Stichworten suchen“ abgeschlossen werden muss, und dadurch das BSF „Rezept auswählen“ aktiviert. Dabei ist es für die Meets-Relation nicht relevant, ob das zweite BSF (im Beispiel die „Rezeptliste“) eine Nebenstelle ist oder nicht. Die Meets-Relation sagt lediglich aus, dass das nachfolgende BSF gestartet und das vorhergehende deaktiviert wird, trifft aber keine Aussagen über das Verhalten des folgenden BSF.

Realisierung der Komposition

Benutzerschnittstellenfragmente, die über die Meets-Relation komponiert werden, zeichnen sich dadurch aus, dass das Beenden eines BSF zum Starten des anderen führt. Somit muss für die Komposition eine Direktive in das Enddialogfragment von BSF1 aufgenommen werden, die das Startdialogfragment von BSF2 aufruft.

8.4.5.3 Overlaps & Overlaped_by-Relation

Die Overlaps&Overlaped_by-Relation beschreibt die Überschneidung in der Durchführung von zwei Benutzerschnittsellenfragmenten. Diese Überschneidung wird realisiert, indem im Ablauf des einen Fragmentes das andere gestartet wird.

(BSF_1 o BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- Eines der Dialogfragmente von BSF_1 startet BSF_2.
- Es existieren in der Ausführung von (BSF_1 o BSF_2) Präsentationsfragmente in BSF1, die aktiv sind, ohne dass ein Präsentationsfragment von BSF_2 aktiv ist, und umgekehrt.
- Es existieren in der Ausführung von (BSF_1 o BSF_2) Präsentationsfragmente von BSF_1 und BSF_2, die gleichzeitig aktiv sind.

Die Overlaps-Relation wird als eine Folge von zwei Benutzerschnittstellenfragmenten interpretiert, die sich partiell in ihrer Ausführung überlappen müssen. Wesentliches Merkmal ist also, dass die Interaktion tatsächlich im anderen Benutzerschnittstellenfragment fortgesetzt wird, was der Semantik der Meets-Relation entspricht.

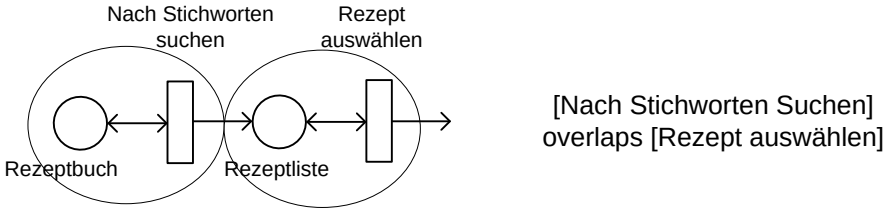


Abbildung 58: Overlaps-Relation im BSF-AIN

Abbildung 58 zeigt eine Overlaps-Relation in einem Ausschnitt aus dem Beispiel in Abbildung 46. Im Startzustand ist nur das BSF „Nach Stichworten suchen“ aktiviert und kann ausgeführt werden. Nachdem das BSF „Nach Stichworten suchen“ ausgeführt wurde, ist sowohl die Stelle „Rezeptbuch“, als auch die Stelle „Rezeptliste“ aktiviert. Folglich können ab dann beide BSF gleichzeitig ausgeführt werden.

Eine Unterscheidung zwischen der Overlaps- und der Overlaped_by-Relation ist nicht möglich, da nicht vorhersehbar ist, welche der beiden Relationen zuerst endet.

Realisierung der Komposition

Benutzerschnittstellenfragmente, die über die Overlaps-Relation komponiert werden, zeichnen sich dadurch aus, dass das Beenden eines BSF zum Starten des anderen führt, wobei das startende BSF weiterhin aktiviert bleibt. Somit muss für eine Komposition nur das Enddialogfragment um zwei Direktiven erweitert werden: Einerseits muss das Startdialogfragment von BSF2 aufgerufen werden, andererseits muss auch das eigene Startdialogfragment (von BSF1) aufgerufen werden, damit das Benutzerschnittstellenfragment selbst wieder verfügbar ist.

8.4.5.4 Contains & During-Relation

Die Contains & During-Relation beschreibt die gleichzeitige Ausführung von zwei Benutzerschnittstellenfragmenten, wobei die Ausführung des einen Benutzerschnittstellenfragmentes die Ausführung des anderen beinhaltet. (BSF_1 c BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- In BSF_1 existiert ein Dialogfragment, das BSF_2 startet. Dieses ist nicht das Startdialogfragment.
- Es existiert ein Dialogfragment, das BSF_2 beendet.
- Wenn ein Präsentationsfragment von BSF_2 aktiv ist, ist auch eines von BSF_1 aktiv.

Die Contains-Relation unterscheidet sich von der Overlaps-Relation dadurch, dass eines der Fragmente durchgehend aktiviert bleibt, während das andere gestartet und wieder beendet wird.

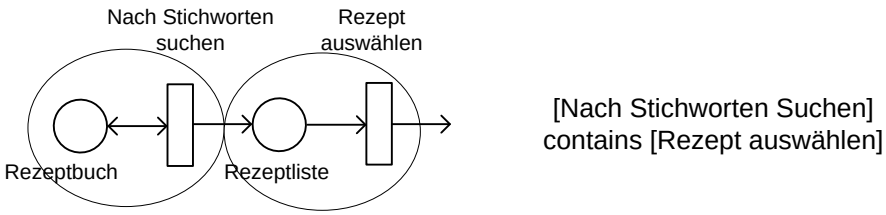


Abbildung 59: Contains-Relation im BSF-AIN

Abbildung 59 zeigt eine Contains-Relation in einem modifizierten Ausschnitt des Beispiels aus Abbildung 46, in welchem die Stelle „Rezeptliste“ nicht mehr Nebenstelle ist. Im Startzustand ist nur das BSF „Nach Stichworten suchen“ aktiviert und kann ausgeführt werden. Nachdem das BSF „Nach Stichworten suchen“ ausgeführt wurde, ist sowohl die Stelle „Rezeptbuch“, als auch die Stelle „Rezeptliste“ aktiviert. Folglich können ab dann beide BSF gleichzeitig ausgeführt werden. Eine erneute Durchführung des BSF „Nach Stichworten suchen“ verändert nicht den Zustand der Markierungen und damit die Sichtbarkeiten der BSF. Wird aber die Transition „Rezept auswählen“ ausgeführt, so wird dadurch das BSF beendet, während „Nach Stichworten suchen“ weiterhin aktiviert bleibt. Damit sind die Anforderungen an eine Contains-Relation erfüllt.

Realisierung der Komposition

Benutzerschnittstellenfragmente, die über die Contains-Relation komponiert werden, zeichnen sich dadurch aus, dass das Beenden eines BSF zum Starten des anderen führt, wobei das startende BSF, wieder aktiviert wird. Der Unterschied zur Overlaps-Relation liegt ausschließlich im Verhalten von BSF_2 begründet, das aber nicht weiter betrachtet wird, da die Komposition immer paarweise durchgeführt wird. Daher wird die Komposition analog zur Overlaps-Relation durchgeführt.

8.4.5.5 Starts & Started_by-Relation

Die Starts & Started_by-Relation beschreibt den gemeinsamen Start der Ausführung zweier Dialogfragmente. Welches der Dialogfragmente zuerst endet, definiert den Unterschied zwischen Starts und Started_by.

(BSF_1 s BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- BSF_1 und BSF_2 werden durch das gleiche Dialogfragment gestartet.

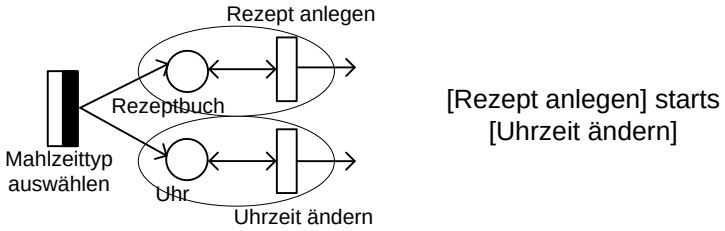


Abbildung 60: Starts-Relation im BSF-AIN

Abbildung 58 zeigt eine Starts-Relation in einem Ausschnitt aus dem Beispiel in Abbildung 46. Nach Ausführung des BSF „Mahlzeit auswählen“ wird sowohl die Stelle „Rezeptbuch“, als auch die Stelle „Uhr“ markiert, und somit die BSF „Rezept anlegen“ und „Uhrzeit ändern“ aktiviert. Folglich können ab dann beide BSF gleichzeitig ausgeführt werden.

Realisierung der Komposition

Benutzerschnittstellenfragmente, die über die Start-Relation komponiert werden, werden gleichzeitig aktiviert. Dies bedeutet für die Komposition, dass ein neues Startdialogfragment aufgebaut wird, welches als Direktive den Aufruf der Startdialogfragmente der beiden Relationen besitzt.

8.4.5.6 Finishes & Finished_by-Relation

Die Finishes & Finished_by-Relation beschreibt das gemeinsame gleichzeitige Ende der Ausführung zweier Dialogfragmente. Welches der Dialogfragmente zuerst gestartet wurde, definiert den Unterschied zwischen Finishes und Finished_by.

(BSF_1 f BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- BSF_1 und BSF_2 werden durch die gleichen Dialogfragmente beendet.

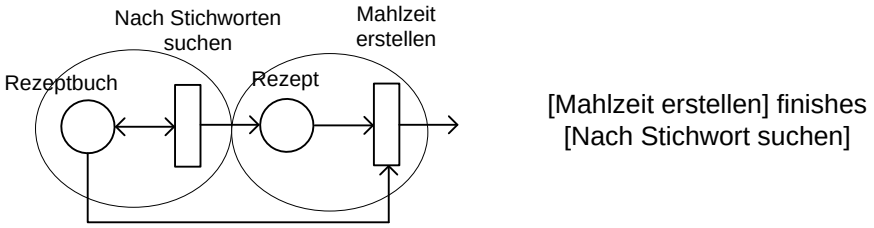


Abbildung 61: Finishes-Relation im BSF-AIN

Abbildung 61 zeigt eine Finishes-Relation in einem modifizierten Ausschnitt des Beispiels aus Abbildung 46, in welchem das BSF „Mahlzeit erstellen“ um die zusätzliche Eingangsstelle „Rezeptbuch “ erweitert wurde. Wie aus dem Beispiel ersichtlich, wird durch Ausführung des BSF Mahlzeit erstellen sowohl die Markierung aus der Stelle „Rezeptbuch“ als auch aus der Stelle „Rezept“ abgezogen. Damit entspricht das Verhalten der Finishes-Relation, da nach der Ausführung von „Mahlzeit erstellen“ auch das BSF „Nach Stichworten suchen“ nicht mehr aktiviert ist.

Realisierung der Komposition

Benutzerschnittstellenfragmente, die über die Finishes-Relation komponiert werden, zeichnen sich dadurch aus, dass beide Benutzerschnittstellenfragmente eine gemeinsame Eingangsstelle haben und die Ausführung von BSF_1 abgeschlossen sein muss, damit BSF_2 ausgeführt werden kann. Folglich muss BSF_2 über zwei Startdialogfragmente verfügen. Die Komposition zweier BSF nach der Finishes-Relation erfordert somit einerseits die Verschmelzung des gemeinsamen PF (vgl. Abschnitt 8.4.5.9) der beiden BSF und weiterhin eine Contains-Relation von BSF1 und BSF2.

8.4.5.7 Equals-Relation

Die Equals-Relation beschreibt die gleichzeitige Ausführung von zwei Benutzerschnittstellenfragmenten. Dies bedeutet, dass beide Benutzerschnittstellenfragmente zur gleichen Zeit starten und zur gleichen Zeit beendet werden.

(BSF_1 = BSF_2) stellt dabei folgende Bedingungen an die Komposition:

- BSF_1 und BSF_2 werden durch das gleiche Dialogfragment gestartet.
- Die Ausführung von BSF_1 und BSF_2 wird durch das gleiche Dialogfragment beendet.
- Invariante: Wenn BSF_1 aktiv ist, so ist auch BSF_2 aktiv und umgekehrt.

Die Equals-Relation stellt Anforderungen an die Komposition, wie sie bereits durch die Starts- und die Finishes Relation beschrieben wurden. Zusätzlich muss aber sichergestellt werden, dass immer beide BSF aktiv sind.

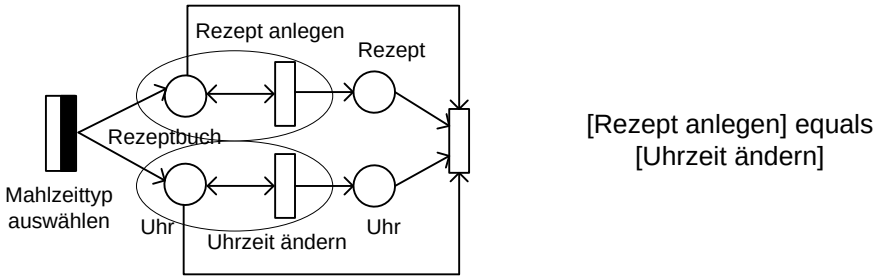


Abbildung 62: Equals-Relation im BSF-AIN

Abbildung 62 zeigt eine Finishes-Relation in einem modifizierten Ausschnitt des Beispiels aus Abbildung 46, in welchem das BSF „Uhrzeit ändern“ die Ausgangsstelle Uhr besitzt. Wie bei der Starts Relation wird auch in diesem Beispiel nach Ausführung des BSF „Mahlzeittyp auswählen“ sowohl das BSF „Rezept anlegen“, als auch das BSF „Uhrzeit ändern“ aktiviert. Durch die Verwendung von Nebenstellen für „Rezeptbuch“ und „Uhr“ wird gewährleistet, dass beide BSF weiterhin aktiviert bleiben. Das gleichzeitige Beenden beider Benutzerschnittstellenfragmente wird durch das Konstrukt der Finishes-Relation (vgl. Abbildung 61) realisiert.

Realisierung der Komposition

Die Equals-Relation zeichnet sich dadurch aus, dass zwei BSF gleichzeitig aktiviert und auch gleichzeitig beendet werden. Während das gleichzeitige Starten in Analogie zur Starts-Relation realisiert werden kann, erfordert das gleichzeitige Beenden die Betrachtung des darauf folgenden BSF (im Folgenden BSF_3 genannt, vgl. Abbildung 62). Eingesetzt wird hier die Finishes-Relation von BSF_1 und BSF_3 und von BSF_2 und BSF_3, wobei BSF_3 über zwei Eingangstellen verfügt und die Ausführung von BSF_3 sowohl BSF_1, als auch BSF_2 beendet.

8.4.5.8 Bedingungen im BSF-AIN

Neben den 7 Relationen der Intervallalgebra muss eine Besonderheit des BSF-AIN betrachtet werden, die zwar keine Auswirkungen auf die Sichtbarkeit der BSF hat, wohl aber für die Komposition von Benutzerschnittstellenfragmenten von Bedeutung ist. Transitionen mit mehr als einer Eingangsstelle definieren Bedingungen, wann eine Transition schalten kann und dementsprechend auch wann bestimmte Interaktionsobjekte im BSF aktiviert werden. Diese Bedingungen müssen bei der Anwendung der Relationen der Kompositionsalgebra beachtet werden.

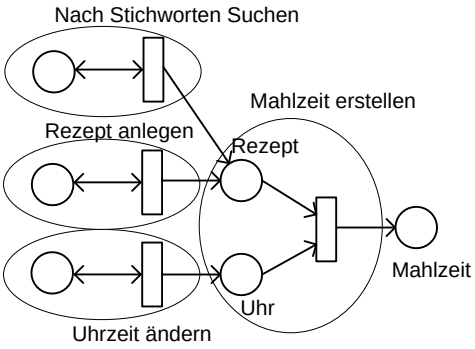


Abbildung 63: Bedingungen im BSF-AIN

Abbildung 63 zeigt einen modifizierten Ausschnitt des Beispiels aus Abbildung 46, in welchem das BSF „Uhrzeit ändern“ die Stelle „Uhr“ aktiviert. Jede der drei BSF „Nach Stichworten Suchen“, „Rezept anlegen“ und „Uhrzeit ändern“ kann das BSF „Mahlzeit erstellen“ aktivieren. Das heißt, dass hier drei Meets-Relationen („Nach Stichworten Suchen“ meets „Mahlzeit erstellen“ usw.) für die Komposition formuliert werden müssen. Das BSF „Mahlzeit erstellen“ wird aber nur dann aktiviert, falls die Bedingung (rechts in Abbildung 63) erfüllt ist, also sobald die Präsentationsfragmente „Rezept“ und „Uhr“ gleichzeitig sichtbar sind.

Realisierung der Komposition

Das Übertragen dieser Bedingungen auf das BSF-AIN wird realisiert, indem die notwendigen Präsentationsfragmente (definiert durch die Eingangsstellen) zur Laufzeit überprüft werden. Jedes Präsentationsfragment definiert seine Sichtbarkeit laut Spezifikation (vgl. Abschnitt 7.2.1.1) über den Parameter Status. Da prinzipiell jedes Startdialogfragment dasjenige sein kann, das die Transition aktiviert, muss jedes Startdialogfragment überprüfen, ob alle notwendigen PF sichtbar sind und erst dann die Transition aktivieren. Für jedes Präsentationsfragment sind folglich entsprechende Direktiven vorgesehen.

8.4.5.9 Alternativen-Relation

Eine weitere Besonderheit (in Bezug auf die Intervallalgebra) ist die Modellierung von Alternativen im BSF-AIN. Alternativen zeichnen sich dadurch aus, dass nur eine der Varianten (bestimmt durch den Benutzer) ausgeführt wird. Dies bedeutet aber für die Untersuchung der Intervallalgebra, dass hier keine Beziehungen zwischen den Benutzerschnittstellenfragmenten vorliegen, und zusätzlich auch nicht eindeutig bestimmbar ist, welches der Benutzerschnittstellenfragmente als nächstes ausgeführt wird. Abbildung 64 visualisiert solche Alternativen am modifizierten Beispiel[41] des Rezeptbuches in Form eines BSF-AIN.

[41] Im Vergleich zum „Rezeptbuch“ im BSF-AIN aus Abbildung 46 ist das „Rezeptbuch“ nicht mehr Nebenstelle die Transitionen „Rezept anlegen“.

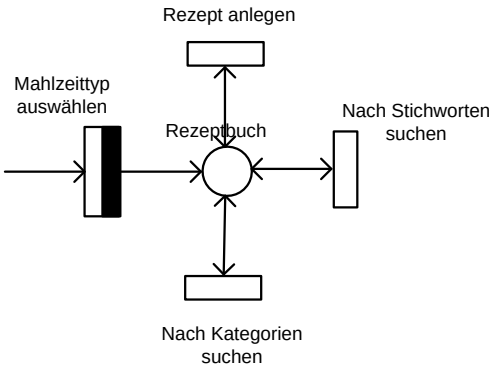


Abbildung 64: Alternativen im BSF-AIN

Dem Beispiel ist zu entnehmen, dass nach Ausführung des BSF „Mahlzeittyp auswählen“ drei mögliche Aktivitäten „Rezept anlegen“, „Nach Stichworten suchen“ und „Nach Kategorien suchen“ folgen können. Auf den ersten Blick handelt es sich aus Sicht der Intervallalgebra um eine „Meets“-Beziehung, bei der nach der Ausführung eines Benutzerschnittstellenfragmentes die Ausführung eines Nächsten folgt. Problematisch ist, dass nicht jede Alternative über die „Meets“-Beziehung mit dem BSF „Mahlzeit auswählen“ verknüpft werden kann. Dies würde nämlich implizieren, dass alle drei Alternativen folgen würden. Im BSF-AIN wurde aber das „exklusive oder“ modelliert.

Eine weitere Besonderheit dieses Konstruktes zeigt ebenfalls das Beispiel in Abbildung 64. Da das Präsentationsfragment „Rezeptbuch“ Eingangsstelle für alle drei BSF ist, sind alle drei gleichzeitig aktiviert, aber nur eines der BSF kann (gleichzeitig) ausgeführt werden. Dies steht im Widerspruch zur sonstigen Interpretation im BSF-AIN. Es wird immer davon ausgegangen, dass falls mehrere BSF zur gleichen Zeit aktiviert sind, diese auch parallel ausgeführt werden können. Dem Beispiel liegt also eine Problemstellung zugrunde, da sich alle BSF ein gemeinsames Präsentationsfragment (modelliert durch die gemeinsame Eingangs- bzw. Nebenstelle) teilen.

Um diese Beziehung abbilden zu können wird eine zusätzliche Relation - die Alternativen-Relation - eingeführt, die nicht Teil der Intervallalgebra ist.

Die Alternativen-Relation beschreibt eine Verknüpfung von Benutzerschnittstellenfragmenten, die alternativ ausgeführt werden. Das Beispiel in Abbildung 64 kann somit folgendermaßen durch die „Meets“-Relation der Intervallalgebra abgebildet werden: „Mahlzeittyp auswählen“ Meets ALT(„Rezept anlegen“, „Nach Stichworten suchen“, „Nach Kategorien suchen“).

ALT(BSF_1,..,BSF_n) stellt dabei folgende Bedingungen an die Komposition:

- Alle BSF teilen sich im BSF-AIN mindestens eine gemeinsame Eingangsstelle. Diese Bedingung gilt auch, wenn es sich bei den Eingangsstellen um Nebenstellen handelt.
- Es erfolgt kein Datenaustausch zwischen den BSF.

Realisierung der Komposition

Benutzerschnittstellenfragmente, die über die Alternativen-Relation komponiert werden, zeichnen sich dadurch aus, dass sie im BSF-AIN mindestens eine gemeinsame Eingangsstelle besitzen. Die Komposition bezieht sich immer auf diese Eingangsstelle. Teilen sich Transitionen im BSF-AIN mehr als eine Stelle, so wird für jede der Stellen die Alternativen-Relation angewendet. In diesem Fall muss zusätzlich durch die „Bedingungen“ (vgl. Abschnitt 8.4.5.8) das richtige Verhalten sichergestellt werden.

Aus Perspektive der BSF erfordert die Komposition über die Alternativen-Relation die Verschmelzung des gemeinsamen PF (Präsentationsfragmentes), das durch die gemeinsame Stelle beschrieben wird. Eine Verschmelzung ist möglich, da die Stelle das gleiche Präsentationsfragment beschreibt, so dass sich die Präsentationsfragmente nur durch ihre Interaktionsobjekte unterscheiden. Aufgabe der Verschmelzung ist daher, ein Präsentationsfragment zu generieren, das alle Interaktionsobjekte der beteiligten Präsentationsfragmente enthält. Basierend auf der Spezifikation eines Präsentationsfragmentes (vgl. Abschnitt 7.2.1.1) muss die Kategorie „Nutzerinteraktionen“ um die Interaktionsobjekte aller Präsentationsfragmente erweitert werden. Zusätzlich muss die in der Kategorie „Spezifikation“ angegebene XAML-Spezifikation um die Interaktionsobjekte erweitert werden. Da Interaktionsobjekte auch „Events“ definieren, auf welche die entsprechenden Dialogfragmente der BSF reagieren, muss hier auf eine Namenskollision hin kontrolliert werden, und gegebenenfalls eine Umbenennung der Events und damit einhergehend eine Umbenennung der Dialogfragmente in einem der BSF durchgeführt werden.

Problematisch (im Zuge der Verschmelzung) ist der Umgang mit „direkten“ Interaktionsobjekten (vgl. Abschnitt 7.2.2.3), da sie gleichzeitig auch für die Darstellung von Inhalten (des PF) verwendet werden. Ein Beispiel für solche Interaktionsobjekte sind Verweislisten im Browser. Diese sind „direkt“, da erst wenn die Interaktion durchgeführt wird, das ausgewählte „Element“ definiert ist. Weiterhin beschreiben diese „Elemente“ auch Inhalte, da jeder Link gleichzeitig auch ein Stück Information bereitstellt. Laut Vorgehen zur Verschmelzung könnte eine Verweisliste folglich zwei Mal im generierten PF enthalten sein, was zu Duplikaten in der Darstellung führen würde. Für die Verschmelzung solcher Interaktionsobjekte ist eine Umwandlung von direkten in indirekte Interaktionsobjekte notwendig. Eine Verweisliste ließe sich beispielsweise einfach über eine Checkliste realisieren, da nur noch ein Button als Interaktionsobjekte übrig bleibt. Solch eine Umwandlung ist aber abhängig von den verwendeten Interaktionsobjekten, die beispielsweise auch „interaktive Landkarten“ oder „interaktive Mindmaps“ sein können. Eine Umwandlung solcher Objekte ist automatisiert nur schwer möglich, so dass eine Verschmelzung ohne die Unterstützung von menschlichen Experten nicht mehr realisiert werden kann.

Neben der Verschmelzung des gemeinsamen PF müssen die Startdialogfragmente der beteiligten BSF verschmolzen werden. Da durch das Aktivieren der gemeinsamen Stelle alle BSF aktiviert werden können, besitzen die BSF auch ein gemeinsames Startdialogfragment, welches alle Direktiven der beteiligten Startdialogfragmente der BSF beinhaltet. Dies beinhaltet insbesondere auch „Aktivierungsbedingungen“ von Transitionen, die mehr als eine Eingangsstelle besitzen.

8.4.5.10 Komposition am Beispiel des Adipositas-Begleiters

Nachdem die Intervallalgebra definiert wurde, muss das BSF-AIN in eine Sequenz aus Relationen zerlegt werden. Da die Relationen jeweils paarweise auf die BSF angewendet werden, sind unterschiedliche Sequenzen möglich. Es muss aber sichergestellt werden, dass jedes BSF (ein- und ausgehende Kanten) im BSF-AIN abgebildet wird. Eine mögliche Strategie sieht vor, sich entlang der Interaktionspfade zu orientieren.

Anhand des Beispiels im BSF-AIN aus Abschnitt 8.4.4 in Abbildung 55 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“ nach der Sequenzialisierung) wird präsentiert, wie die einzelnen Relationen der Kompositionsalgebra eingesetzt werden können.

ALTERNATIVEN:

Rezeptbuch = ALT ([Rezept anlegen], [Nach Stichworten suchen], [Nach Kategorien suchen])

Rezept = ALT ([Zu den Favoriten hinzufügen], [weiter])

Uhr = ALT ([Uhrzeit ändern], [Mahlzeit erstellen])

Mahlzeit = ALT ([Mahlzeit betrachten], [Mahlzeit zum Plan hinzufügen]

NETZ:

[Mahlzeittyp auswählen] OVERLAPS Rezeptbuch

[Rezept anlegen] OVERLAPS Rezept

[Nach Stichworten suchen] OVERLAPS ([Rezept auswählen] OR Rezept])

[Nach Kategorien suchen] OVERLAPS ([Rezept auswählen] OR Rezept])

[Rezept auswählen] MEETS Rezept

[Weiter] MEETS Uhr

[Mahlzeit erstellen] MEETs Mahlzeit

NEBENSTELLEN:

[Zu den Favoriten hinzufügen] MEETS [Zu den Favoriten hinzufügen]

[Uhrzeit ändern] MEETS [Uhrzeit ändern]

[Mahlzeit betrachten] MEETS [Mahlzeit betrachten]

Dargestellt sind die Relationen, die aus dem Beispiel in Abbildung 55 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“ nach der Sequenzialisierung) identifiziert wurden.

Um die Übersicht zu wahren, werden in einem ersten Schritt die Alternativen-Relationen ausgeführt. Weiterhin wird dem dadurch generierten BSF ein eigener Name zugeordnet. Um die Lesbarkeit zu erhöhen, wurde jeweils der Name der Stelle gewählt, über welche die Alternativen-Relation gebildet wird.

Im nächsten Schritt wurde das BSF-AIN beginnend mit der Transition „Mahlzeittyp auswählen“ (Starttransition) abgebildet. Jede Zeile beschreibt dabei eine Relation, wobei Transitionen durch eckige und Stellen durch geschweifte Klammern gekennzeichnet werden, und zusätzlich „and“-, „or“- und „if“-Konstrukte für die Beschreibung von Bedingungen verwendet wurden.

Die Darstellung der Relationen folgt der Syntax „<Transition$_1$> <Relation> <Transition$_2$>“. Einzige Ausnahme bildet die Alternativen Relation mit der Syntax „ALT{<PF>}(<Transition$_1$>, <Transition$_2$>,...,<Transition$_n$>)“. Auf eine Angabe des Präsentationsfragmentes kann verzichtet werden, wenn die angegebenen Transitionen nur über eine gemeinsame Eingangsstelle verfügen.

Im letzten Schritt erfolgt die Behandlung der Transitionen, die als einzige Stelle mit einer „Nebenstelle“ verknüpft sind. Realisiert werden solche Transitionen, indem sie durch eine „MEETS“-Verknüpfung mit sich selbst komponiert werden. Dadurch wird sichergestellt, dass die Transition, nachdem sie ausgeführt wurde, direkt wieder schalten kann.

Die Realisierung der Komposition der Benutzerschnittstellenfragmente erfordert die Ausführung der Anweisungen, die den einzelnen Relationen zugeordnet wurden.

8.5 Zusammenfassung und abschließende Betrachtung

Zu Beginn des Kapitels wurde folgende Aufgabenstellung formuliert (vgl. Abschnitt 8.1):

> *Gegeben seien eine Interaktionsbeschreibung in Form eines Abstract Interaction Nets (definiert in Abschnitt 5.3), eine „Domain Theory“ (definiert in Abschnitt 7.3) bestehend aus wieder verwendbaren Benutzerschnittstellenfragmenten (definiert in Abschnitt 7.2.1) und ein Nutzungsprofil (definiert in Abschnitt 6.3).*
>
> *Aufgabenstellung ist es nun basierend auf den Vorgaben der Interaktionsbeschreibung durch Komposition der Benutzerschnittstellenfragmente der „Domain Theory“ eine ausführbare Benutzerschnittstelle (definiert in Abschnitt 7.2.2.2) zu generieren, welche den durch das Nutzungsprofil definierten Anforderungen bestmöglich genügt.*

Im Rahmen des Kapitels konnte gezeigt werden, dass eine Generierung solch einer Benutzerschnittstelle weitestgehend automatisiert erfolgen kann.

Verwendet wurde dabei ein zweistufiges Verfahren. In einem ersten Schritt wurden basierend auf den Anforderungen, die durch das Nutzungsprofil definiert sind, Benutzerschnittstellenfragmente identifiziert, die in einem zweiten Schritt komponiert wurden.

Die vorgestellte Methodik zur Identifikation der notwendigen Benutzerschnittstellenfragmente bediente sich primär des OM-Kataloges als Wissensbasis und nutzt die schrittweise Dekomposition von Objekt- und Interaktionsmetaphern zu AINs, um unterschiedliche Abbildungen von Benutzerschnittstellenfragmenten auf die Interaktionsbeschreibung zu generieren. Mit Hilfe einer Bewertungsfunktion wird es möglich, jede einzelne Abbildungen zu gewichten und eine auf das Nutzungsprofil hin optimierte Lösungsmenge zu bestimmen. Die Qualität der resultierenden Ergebnisse dieses Ansatzes ist dabei erheblich von der verfügbaren Wissensbasis (OM-Katalog und „Domain Theory“) abhängig. Neben den Möglichkeiten zur Dekomposition, die vom strukturierten Aufbau des OM-Katalogs abhängig ist, müssen insbesondere für die Bewertungsfunktion die Elemente der „Domain Theory“ über aussagekräftige Nutzungskontextvektoren verfügen, welche aber nicht vollständig automatisiert ermittelt werden können.

Das Ergebnis des ersten Generierungsschrittes ist ein sogenanntes BSF-AIN, das sich der Syntax eines AINs bedient, für das aber eine Abbildung auf Benutzerschnittstellenfragmente der „Domain Theory“ angegeben werden kann. Das BSF-AINs modelliert die Beziehungen der Benutzerschnittstellenfragmente untereinander und spezifiziert somit wie diese zu komponieren sind.

Die eigentliche Komposition erfolgt in zwei Phasen. In der ersten Phase werden die im BSF-AIN modellierten Beziehungen zwischen den Benutzerschnittstellenfragmenten untersucht und überprüft, ob sich diese auf das im Nutzungsprofil gegebene Layoutpattern abbilden lassen. Ist dies nicht der Fall, so wird versucht, die Beziehungen dahingehend zu modifizieren, dass die Möglichkeit zur parallelen Durchführung von Interaktionen beschränkt wird, indem sie (falls notwendig) in eine Sequenz überführt werden. Der hierfür verwendete Algorithmus nutzt einen „Brute Force“-Ansatz der Methoden zur Modifikation auf das gegebene BSF-AIN anwendet und erst im Anschluss überprüft, ob das Ergebnis noch valide ist und den Anforderungen des ursprünglichen BSF-AINs entspricht. Hierbei muss festgestellt werden, dass zwar automatisiert mögliche Lösungen ermittelt werden können, aber eine automatisierte Bewertung der Lösungsmenge nur eingeschränkt möglich ist und daher gegebenenfalls auf menschliche Experten zurückgegriffen werden muss.

Die zweite Phase realisiert die Komposition der Benutzerschnittstellenfragmente indem eine Intervallalgebra aufgebaut wird, welche in der Lage ist, die im BSF-AIN modellierten Beziehungen in einzelne Relationen von Benutzerschnittstellenfragmenten zu überführen. Jeder Relation wird dabei eine Kompositionsvorschrift zugeordnet, welche sich auf die formale Spezifikation eines Benutzerschnittstellenfragmentes anwenden lässt. Indem das BSF-AIN in eine Menge aus Relationen überführt wird, wird definiert, wie die Benutzerschnittstellenfragmente zu komponieren sind.

Zusammenfassend kann festgehalten werden, dass die Komposition weitgehend automatisiert erfolgen kann. An den ausgezeichneten Stellen im Kompositionsprozess kann es aber notwendig sein, dass eine Nachbearbeitung durch menschliche Experten erfolgen muss. Der bausteinorientierter Ansatz ermöglicht es gewählte Lösungswege und manuell durchgeführte Modifikationen weitgehend zu sichern und im Fall eines wiederholten Durchlaufens des Kompositionsprozesses direkt verfügbar zu machen.

Diese Aussage lässt sich auf den ganzen Generierungsprozess (von der Modellierung bis zur Komposition) dahingehend übertragen, dass eine ausreichend große Wissensmenge verfügbar sein muss, um effizient generieren zu können. Dies bedeutet aber, dass insbesondere die ersten Realisierungen von Benutzerschnittstellen mit erheblichem Mehraufwand verbunden sind, um die notwendige Datenmenge zu erzeugen. Die in dieser Arbeit geleiteten Vorarbeiten können nur die ersten Fundamente für solch eine Wissensbasis liefern, die erst mit zunehmender Nutzung anwachsen und kontinuierlich nutzbringender wird und den Generierungsprozess effizienter macht.

9 Evaluation und Bewertung des Verfahrens am Beispiel des Adipositas-Begleiters

In den vorhergehenden Kapiteln wurden Modelle und Methoden für eine automatisierte Generierung von Benutzerschnittstellen erarbeitet. Weiterhin wurde ein Verfahren definiert, das es ermöglicht, Benutzerschnittstellen basierend auf den erarbeiteten Modellen und Methoden durch Komposition zu erzeugen.

Auch wenn bereits in weiten Teilen der Arbeit versucht wurde, diese Modelle, Methoden und auch Zwischenergebnisse im Generierungsprozess anhand des Beispiels des Adipositas-Begleiters zu veranschaulichen und zu evaluieren, fehlt weiterhin eine komprimierte und durchgehende Darstellung des Generierungsverfahrens über die einzelnen Schritte (einschließlich der Modellierung) hinweg.

Ziel dieses Kapitels ist die Evaluation der Methodik und sowohl der praktische Nachweis der Funktionsweise als auch eine Bewertung bezüglich der Zielsetzung der Effizienzsteigerung für die Benutzerschnittstellengenerierung. Durchgeführt wird die Evaluation durch die praktische Generierung von Benutzerschnittstellen für die drei in Abschnitt 6.4.4 (vgl. Tabelle 8) eingeführten Nutzungskontexte („Youngster unterwegs mit dem PDA“, „Best Ager zuhause am Fernseher“ und „Mid Ager bei der Arbeit am PC“). Diese Nutzungskontexte definieren Anforderungen an Benutzerschnittstellen für unterschiedliche Nutzer, Endgeräte und Umgebungen. Die generierte Benutzerschnittstelle muss diesen Anforderungen bestmöglich gerecht werden. Dabei definieren das Nutzungskontextmodell, die „Domain Theory“ und der Objektmetaphern-Katalog die für die Generierung zur Verfügung stehende Wissensbasis.

Als Interaktionsbeschreibung für die Evaluation wurde ein Teil der Benutzerschnittstelle des Adipositas-Begleiters herangezogen. Als Ausschnitt wurde die Realisierung des „Ernährungstagebuches“ und hier der in der Arbeit eingeführte Teil „Mahlzeit erstellen“ ausgewählt. Dieses Beispiel wurde bereits im Verlauf der gesamten Arbeit verwendet, um Konzepte der Generierung von Benutzerschnittstellen zu veranschaulichen und um praxisnahe Beispiele für deren Umsetzung zu geben. Diese Wiederholung wurde bewusst gewählt, um dem Leser im Bedarfsfall einen detaillierten und auch anwendungsfallbezogenen Rückblick auf die verwendeten Methoden zu ermöglichen, die in den Kapiteln 5 bis 8 zu finden sind.

In den folgenden Abschnitten wird der Generierungsprozess für die drei Nutzungsprofile eingehend beschrieben. Als erster Nutzungskontext wird „Youngster unterwegs mit dem PDA“ ausgewählt. Dieser Nutzungskontext dient als Beispiel für die Veranschaulichung des Kompositionsprozesses in Kapitel 8. Bei der Beschreibung des Generierungsprozesses für die anderen beiden Nutzungskontexte werden insbesondere Unterschiede im Generierungsprozess aufgezeigt und diskutiert. Das Ergebnis der Generierung wird für alle drei Nutzungskontexte zusätzlich anhand von realen Benutzerschnittstellen visualisiert, und der Bezug zu den Modellen, aus denen diese generiert wurden, hergestellt. Abschließend erfolgt eine Diskussion der Ergebnisse der Evaluation und der generierten Benutzerschnittstellen.

9.1 „Youngster unterwegs mit dem PDA“

9.1.1 Modellierung der Interaktionsbeschreibung

Der Konzeption des Generierungsprozesses (vgl. Abschnitt 8.1) folgend, definieren die Interaktionsbeschreibung und der Nutzungskontext die Ausgangslage für die Generierung der Benutzerschnittstelle. Die Interaktionsbeschreibung erfolgt mit Hilfe der in Abschnitt 5.3 eingeführten graphischen Modellierungssprache AIN (Abstract Interaction Net). Diese Modellierungssprache beschreibt Interaktionen des Anwenders mit Hilfe von Objekt- und Interaktionsmetaphern, ohne Aussagen über die Realisierung der konkreten Benutzerschnittstellen, der Fähigkeiten des Anwenders oder der Umgebung der Nutzung zu treffen.

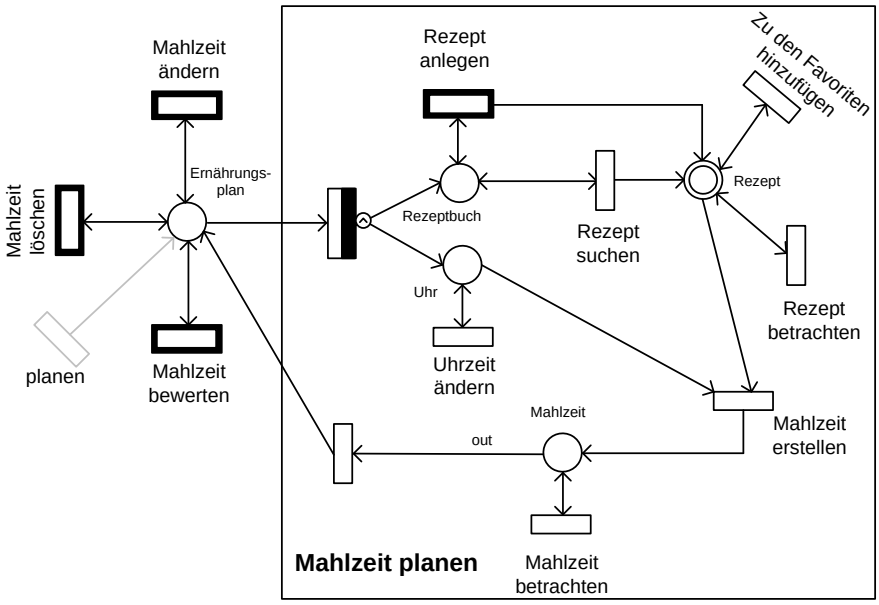


Abbildung 65: AIN für die Interaktion „Mahlzeit planen“

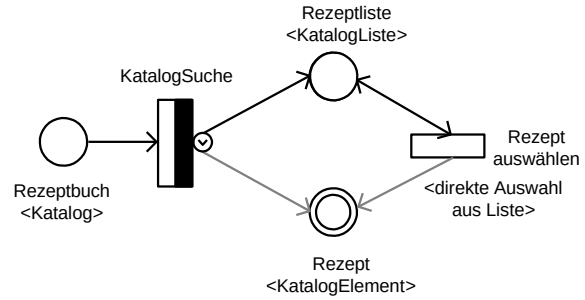
Abbildung 65 veranschaulicht ein Beispiel für solch eine Interaktionsbeschreibung. Dieses Beispiel wurde in Abschnitt 5.6 eingeführt und dort eingehend beschrieben.

Damit solch ein AIN für eine automatisierte Generierung verwendet werden kann, muss eine Beziehung zu den Elementen des „Domänenspezifischen Baukastens für die Benutzerschnittstellengenerierung“ (vgl. Abschnitt 8.1) existieren. Hergestellt wird diese Beziehung dadurch, dass alle Elemente (Stellen und Transitionen) der Interaktionsbeschreibung im Objektmetaphern-Katalog (im Folgenden kurz OM-Katalog genannt) beschrieben werden. Das bedeutet aber auch, dass die Modellierung der Interaktionsbeschreibung mit einer Erweiterung des OM-Kataloges einhergeht. Werden Objekt- oder Interaktionsmetaphern in der Interaktionsbeschreibung verwendet, die nicht Teil des OM-Kataloges sind, so müssen diese in den Katalog aufgenommen und dort spezifiziert werden.

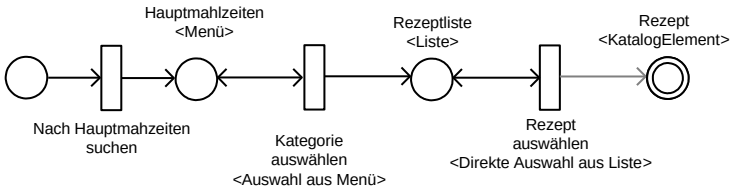
Realisiert wird die Spezifikation neuer Elemente, indem neue Objekt- und Interaktionsmetaphern in den OM-Katalog aufgenommen werden. Neue Interaktionsmetaphern müssen weiterhin durch ein AIN und/oder durch die Angabe eines Benutzerschnittstellenfragmentes, das diese Interaktionsmetapher realisiert, beschrieben werden. Zielsetzung der Beschreibung von Interaktionsmetaphern ist es, diese möglichst durch existierende Elemente des OM-Kataloges zu beschreiben. Solch eine Beschreibung erfolgt durch ein AIN, für das gilt, dass alle für die Beschreibung verwendeten Elemente auch Teil des OM-Kataloges sein müssen. Bei der Spezifikation einer Interaktionsmetapher durch ein AIN können Erweiterungen des OM-Kataloges notwendig werden, falls Objekt- oder Interaktionsmetaphern in diesem AIN verwendet werden, die nicht Teil des OM-Kataloges sind.

Angenommen die Interaktionsmetapher „Rezept suchen“ wurde im Rahmen der Modellierung der Interaktionsbeschreibung als neue Interaktionsmetapher eingeführt, so kann diese folgendermaßen (vgl. Metapher „Rezeptbuch“ in Anhang A) im OM-Katalog spezifiziert werden:

ObjektMetapher: [RezeptBuch]
Abstrakt: {}<->RezeptSuchen->{[Rezept]}
Alternative: KategorieSuche
Variante: SuchenImKatalog
AIN: SucheImKatalog_AIN



BSF:{KatalogSuche.bsf}
Variante: EinfacheSucheInKategorien
AIN: EinfacheSucheInKategorien_AIN



BSF: {}

Tabelle 14: Ausschnitt der Metapher „Rezeptbuch“ aus dem OM-Katalog

Wie aus dem Ausschnitt ersichtlich ist, wurde „RezeptSuchen“ als „Abstrakte Interaktion“ in den OM-Katalog aufgenommen. Dies impliziert, dass Alternativen und Varianten für die Beschreibung der Interaktionsmetapher verwendet werden können. Dem Ausschnitt in Tabelle 14 ist zu entnehmen, dass nur eine Alternative „KategorieSuche“ spezifiziert wurde, die aber durch zwei Varianten („SucheImKatalog“ und „EinfacheSucheInKategorien“) realisiert werden kann.

Die erste Variante „SucheImKatalog“ bedient sich eines „Kataloges“ um die Rezeptsuche zu beschreiben, während die zweite Variante („EinfacheSucheImKatalog“) einfache Kategorielisten verwendet, über die der Anwender navigieren kann. Die erste Variante spezifiziert darüber hinaus noch ein Benutzerschnittstellenfragment „KatalogSuche.bsf“, das diese Variante realisiert.

9.1.2 Modellierung des Nutzungskontextes und Generierung des Nutzungsprofils

Neben der Interaktionsbeschreibung ist weiterhin die Definition eines Nutzungsprofils erforderlich, in dem die Anforderungen an die zu generierende Benutzerschnittstelle auf einer sehr detaillierten Ebene in Form von „Parameter: Wert“-Paaren spezifiziert sind. Aufgrund des Umfangs wird das Nutzungsprofil entsprechend dem Generierungsprozess nicht manuell erstellt, sondern aus einem Nutzungskontext heraus erzeugt.

	Umgebung	Nutzer	Endgerät
Default-Profil	Unterwegs	technikaffiner Nutzer	Apple iPhone 3G
Kontext-Element	- Lichtverhältnisse: + - Personen im Umfeld: - - Ablenkung: +	- Sehvermögen: ++ - Motorik +	

Tabelle 15: Nutzungskontext „Youngster unterwegs mit dem PDA“

Tabelle 15 visualisiert die Beschreibung eines Nutzungskontextes, wie es in Abschnitt 6.4 eingeführt wurde. Definiert wird dieser durch die Angabe von jeweils einem Default-Profil für die drei Nutzungskontextkomponenten (Umgebung, Nutzer, Endgerät), wobei jede dieser drei Komponenten durch zusätzliche Kontextelemente verfeinert werden kann. Beschrieben werden Kontextelemente und auch Default-Profile im Rahmen eines Kontextmodells, das dem Modellierer als Werkzeug für die effiziente Definition von Nutzungskontexten zur Verfügung steht.

In Abschnitt 6.4 wurde ein solches Kontextmodell vorgestellt, welches Default-Profile und Kontextelemente für die Anwendungsdomäne des Adipositas-Begleiters bereitstellt. Der in Tabelle 15 visualisierte Nutzungskontext „Youngster unterwegs mit dem PDA“ ist folgendermaßen definiert (vgl. Abschnitt 6.4):

- **Umgebung:** Unterwegs
  Beschreibung: „Unterwegs“ beschreibt ein Umfeld, in dem der Benutzer mobil ist, wie beispielsweise als Fußgänger, Bahnfahrer aber auch Autofahrer. Der Benutzer ist weitgehend abgelenkt und zumeist von Menschen umgeben.

Kontextelemente:

- Störung durch Umgebungslärm (+)
- Präferenzen für eine sequentielle Aufgabendurchführung. (++)
- Präferenzen für beschriftete Interaktionsobjekte. (++)
- Präferenzen für komplexe, interaktive Interaktionsobjekte und Interaktionen. (-)
- Präferenz für die Nutzung von dynamischen Fenstern. (--)

- **Nutzer:** Technikaffine Nutzer
  Beschreibung: Menschen, die Gefallen an dem Umgang mit interaktiver Software haben und Interesse entwickeln, die Möglichkeiten zu erproben. Sie zeichnen sich durch einen spielerischen Umgang mit dem Softwaresystem aus und die Bereitschaft, durch „Ausprobieren“ die Anwendung zu erkunden.
  Kontextelemente:
  - Präferenzen für symbolische und graphische Darstellungen. (++)
  - Präferenzen für die Nutzung von multimedialen Inhalten. (+)
  - Präferenzen für eine flexible Aufgabendurchführung. (++)
  - Präferenzen für komplexe, interaktive Interaktionsobjekte und Interaktionen. (++)
  - Präferenzen für hohe Informationsdichte auf der Benutzerschnittstelle. (++)
  - Präferenz für die Nutzung von dynamischen Fenstern. (+)

- **Endgerät:** Apple iPhone 3G
  Beschreibung: Die Kontextkomponente Endgerät nimmt eine Sonderstellung im Rahmen der Kontextmodellierung ein, da sie nicht durch Kontextelemente beschrieben wird, sondern bereits existierende Beschreibungen von Endgeräteprofilen nutzt, wie sie beispielsweise durch UAProf (vgl. [OMA06]) gegeben sind. In Abschnitt 6.3.4 wird eine Erweiterung des „Hardware Profiles“ des UAProf vorgestellt, die detaillierter Auskunft über physische Ein- und Ausgabemöglichkeiten des verwendeten Endgerätes gibt. Jedem Endgerät wird darüber hinaus ein Layoutpattern (vgl. Abschnitt 6.3.5) zugeordnet, das die Benutzerschnittstelle in eine Reihe von Bereichen zerlegt, die für die Darstellung von Benutzerschnittstellenfragmenten zur Verfügung stehen.

Bis auf das Endgerät, das direkt Parameter eines Nutzungskontextes spezifiziert, wird jede Nutzungskontextkomponente (Umgebung, Nutzer, Endgerät) durch eine Reihe von Kontextelementen beschrieben, die durch das jeweilige Default-Profil und gegebenenfalls durch zusätzliche Kontextelemente angegeben werden.

In Abschnitt 6.4 wurde gezeigt, wie aus Nutzungskontexten automatisiert Nutzungsprofile ermittelt werden können. In Anhang B findet sich die Spezifikation des generierte Nutzungsprofil für „Youngster unterwegs mit dem PDA“.

9.1.3 Generierung des BSF-AINs

Das Nutzungsprofil und die Interaktionsbeschreibung definieren die beiden Eingaben für den automatisierten Generierungsprozess. Ziel des Generierungsprozesses ist in einem ersten Schritt die Interaktionsbeschreibung auf die Benutzerschnittstellenfragmente der „Domain Theory“ abzubilden, und in einem zweiten Schritt die Komposition der ermittelten Benutzerschnittstellenfragmente zu einer ausführbaren Benutzerschnittstelle durchzuführen.

Die Ermittlung solch einer Abbildung erfolgt mit Hilfe des in Abschnitt 8.2.2 vorgestellten Algorithmus, der die Interaktionsbeschreibung und das Nutzungsprofil als Input nimmt und auf Basis des OM-Kataloges und der „Domain Theory“ ein BSF-AIN generiert.

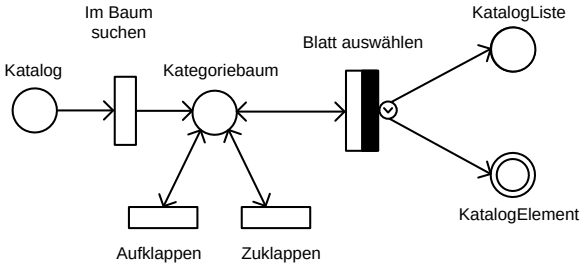
Der Algorithmus nutzt den OM-Katalog, um für jede Interaktionsmetapher in der Interaktionsbeschreibung ein für das Nutzungsprofil passendes Benutzerschnittstellenfragment oder ein AIN zu identifizieren, welches diese Interaktionsmetapher realisiert. Vereinfacht dargestellt, berechnet der Algorithmus unterschiedliche Varianten für die Realisierung einer Interaktionsmetapher und vergleicht, ob ein BSF der „Domain Theory“ oder eines der beschreibenden AINs im OM-Katalog bezogen auf das Nutzungsprofil besser geeignet ist. Dabei können auch neue AINs gebildet werden, die durch die Strukturierungen in „Abstrakte Interaktion“, „Alternativen“ und „Varianten“ (vgl. Struktur des OM-Kataloges in Abschnitt 5.4) beschrieben werden. In jedem Schritt wird überprüft, ob für die angegebenen Interaktionsmetaphern im OM-Katalog Realisierungen in Form von Benutzerschnittstellenfragmenten gefunden werden können. Nur Benutzerschnittstellenfragmente können mit Hilfe der in Abschnitt 8.2.3 angegebenen Bewertungsfunktion auch bewertet und somit miteinander verglichen werden. Diese Bewertung findet auf Basis der in der „Domain Theory“ verzeichneten Nutzungskontextvektoren statt. Jedes Benutzerschnittstellenfragment verfügt über solch einen Nutzungskontextvektor, welcher das Benutzerschnittstellenfragment in einer Form beschreibt, die eine Bewertung gegenüber dem Nutzungsprofil ermöglicht.

Wie die unterschiedlichen Varianten für die Beschreibung einer Interaktionsmetapher gebildet werden, wurde bereits anhand der Abbildung 44 in Abschnitt 8.3 visualisiert und diskutiert. Im Rahmen der an dieser Stelle durchgeführten Evaluation wird darüber hinaus aufgezeigt, welche Varianten für das konkrete Nutzungsprofil „Youngster unterwegs mit dem PDA“ gebildet werden, und wie durch den Einsatz der Bewertungsfunktion passende Benutzerschnittstellenfragmente ermittelt werden.

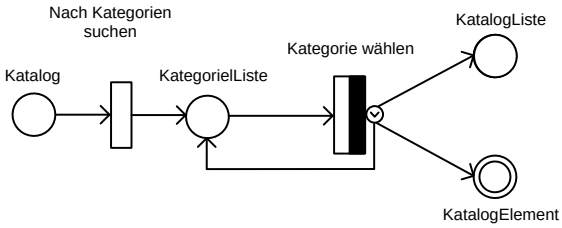
Ausgangslage für diese Betrachtung ist die Interaktionsbeschreibung (wie sie im vorhergehenden Abschnitt eingeführt wurde) und der OM-Katalog. Im vorhergehenden Abschnitt (vgl. Tabelle 14) wurde weiterhin dargestellt, wie die Interaktionsmetapher „Rezept suchen“ im Rahmen der Erstellung der Interaktionsbeschreibung im OM-Katalog spezifiziert ist. Dieser Definition entsprechend kann eine Variante

der „Rezeptsuche“ als Suche in einem Katalog (vgl. Variante SucheImKatalog_AIN in Tabelle 14) aufgefasst werden.

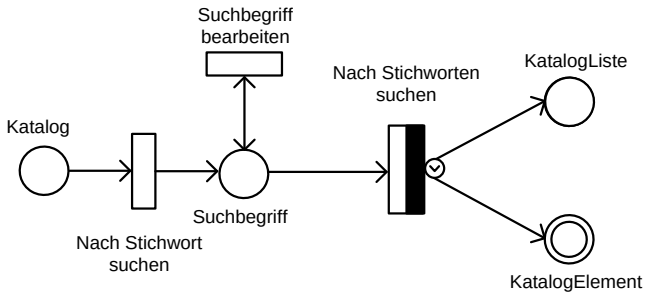
ObjektMetapher: Katalog
Abstrakt: {}<->KatalogSuche->{[KatalogListe]v[KatalogElement]}
Alternative: SucheInKategorien
Variante: SucheImKategoriebaum
AIN: SucheImKategoriebaum _AIN



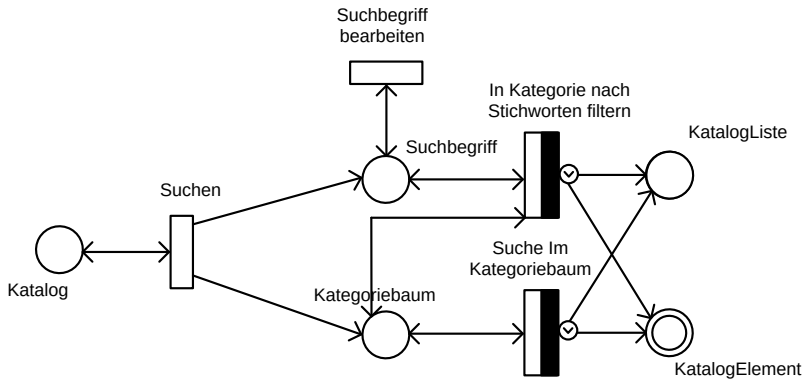
BSF: {SimpleTree.bsf, TreeView.bsf, TreeViewCIPhone.bsf}
Variante: ZweistufigeSuche
AIN: ZweistufigeSuche_AIN



BSF: {KatListe.bsf}
Alternative: StichwortSuche
Variante: 1FeldFormular
AIN: 1FeldForm_AIN



BSF: {SuchForm.bsf}
Alternative: Kat&StichwortSuche
Variante: ParalleleKat&Stichwortsuche
AIN: ParalleleKat&1FeldForm_AIN



BSF: {}

Tabelle 16: Ausschnitt aus dem OM-Katalog: Interaktionsmetapher „Katalogsuche“

In Tabelle 16 ist daher ergänzend zu dem Ausschnitt aus dem OM-Katalog (in Tabelle 14) ein weiterer Ausschnitt des OM-Kataloges abgebildet, in dem beschrieben wird, wie die Interaktionsmetapher „KatalogSuche“ spezifiziert wurde.

Die Interaktionsmetapher „Rezept suchen“ hat als Eingangsstelle die Objektmetapher „Katalog“ und als Ergebnis entweder eine „KatalogListe“ oder ein „KatalogElement“. „KatalogSuche“ selbst ist eine „Abstrakte Interaktion“, die über drei Alternativen verfügt:

- „SucheInKategorien“
  Beschreibt die Realisierungen der Interaktionsmetapher durch die Navigation in Kategorien. Die Varianten reichen dabei von komplexen baumartigen Strukturen bis zu einfachen Kategorielisten.
- „StichwortSuche“
  Beschreibt die Realisierung der Interaktionsmetapher als eine Suche auf Basis von Freitexteingaben.
- „Kat&StichwortSuche“
  Beschreibt die Realisierung der Interaktionsmetapher als Konglomerat der beiden vorhergehenden Alternativen, wobei der Benutzer beide Methoden in Kombination nutzen kann. Das heißt, er kann beispielsweise eine Kategorie auswählen und in dieser nach bestimmten Stichworten suchen, die er über ein Freitextfeld definiert. Hierzu stehen ihm gleichzeitig eine Kategorieliste und ein Suchfeld für die Filterung zur Verfügung.

Mit Hilfe der Ausschnitte (in Tabelle 14 und Tabelle 16) aus dem OM-Katalog kann die Funktionsweise des Algorithmus zur Generierung des BSF-AINs schrittweise evaluiert werden. Abbildung 66 visualisiert dieses Verfahren anhand des Beispiels „Rezept suchen“ in Form eines Baumes. Die einzelnen Knoten repräsentieren AINs, BSF oder Interaktionsmetaphern (wobei zwischen Abstrakten, Alternativen und Varianten unterschieden wird). Die Kanten beschreiben mögliche Realisierungen, die anhand des OM-Kataloges ermittelt werden. Ein Knoten ist dabei folgendermaßen beschriftet:

- Typ

  Der Typ beschreibt, ob es sich um eine Interaktionsmetapher, ein AIN oder ein BSF handelt.

- Name

  Gibt den Namen des Elementes im OM-Katalog an. Sollte mehr als ein BSF für die Realisierung im OM-Katalog angegeben sein, so sind diese in einem Knoten zusammengefasst.

- Wert (in rot)

  Gibt den Wert an, den die Bewertungsfunktion unter Berücksichtigung des Nutzungsprofils für die angegebene Realisierung ermittelt hat.

- Auswahl (optional)

  Die gelbe Umrandung um einen Knoten gibt an, dass dieser im Rahmen des Algorithmus als bestes Ergebnis ausgewählt wurde.

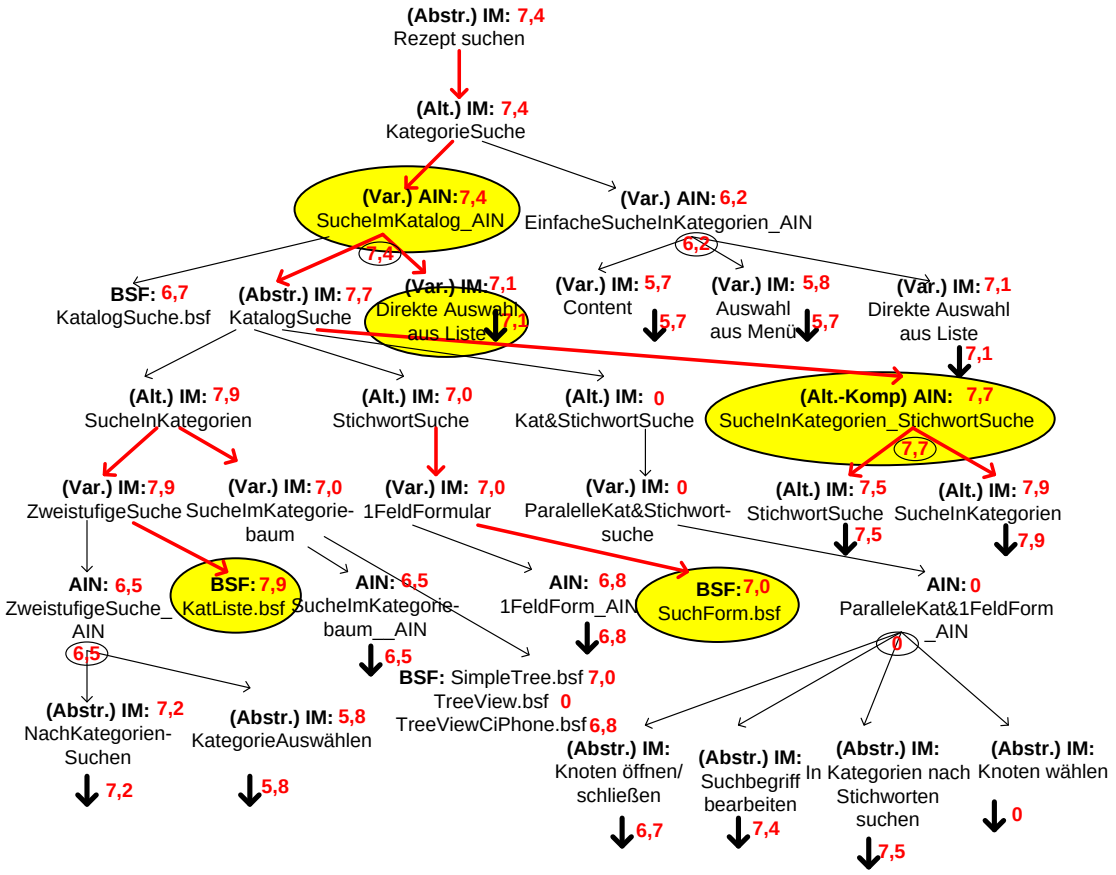


Abbildung 66: Ermittlung des BSF-AINs – „Youngster unterwegs mit den PDA“

Die Wurzel des Baumes bildet die zu untersuchende „Abstrakte Interaktionsmetapher“ „Rezept suchen“. Laut OM-Katalog (vgl. Ausschnitt aus dem OM-Katalog in Tabelle 14) gibt es nur eine Alternative („Kategoriesuche“), die durch zwei Varianten „SucheImKatalog“ und „EinfacheSucheInKategorien“ realisiert werden kann. Die beiden Kinder der „Alternativen Interaktion“ „KategorieSuche“ im Baum sind die jeweiligen AINs. Betrachtet man den Knoten „SucheImKatalog_AIN“, so existiert laut OM-Katalog das

BSF „KatalogSuche.bsf“, das eine Realisierung dieser Variante ermöglicht. Weiterhin kann natürlich auch das AIN eine mögliche Realisierung beschreiben. Um diese genauer zu untersuchen, müssen die einzelnen Interaktionsmetaphern („KatalogSuche“ und „Direkte Auswahl aus Liste“) des AIN analysiert werden. Im Baum ist die Analyse eines AINs dadurch gekennzeichnet, dass die ausgehenden Kanten in einem Punkt starten und zusätzlich über eine Beschriftung (eingekreiste Zahl) verfügen, welche das Ergebnis der einzelnen Bewertungen zusammenfasst. Dies bedeutet, dass für die Bewertungen die einzelnen Interaktionsmetaphern zusammen betrachtet werden müssen. „KatalogSuche.bsf“ hingegen bildet dazu eine optionale Realisierung und verfügt über eine eigenständige Kante.

Eine weitere Besonderheit zeigt sich in der Analyse von „Abstrakten Interaktionen“, die über unterschiedliche Alternativen verfügen. „KatalogSuche“ beispielsweise verfügt laut OM-Katalog (vgl. Tabelle 16) über die drei Alternativen „SucheInKategorien“, „StichwortSuche“ und „Kat&StichwortSuche“. Laut dem Algorithmus zur Ermittlung des BSF-AINs muss in einem ersten Schritt jede der Alternativen einzeln untersucht werden. Zusätzlich muss (abhängig vom Nutzungsprofil) gegebenenfalls ein neues AIN generiert werden, das dem Benutzer den Zugriff auf unterschiedliche Alternativen in der Nutzung der Benutzerschnittstelle ermöglicht. Für das angegebene Nutzungsprofil „Youngster unterwegs mit dem PDA“ ermittelt der Algorithmus genau solch einen Bedarf und generiert das AIN „SucheInKategorien_StichwortSuche“. Im Baum ist solch ein neu generiertes AIN durch den Typ „(Alt.-Komp)“ gekennzeichnet. Für die Entscheidung, ob mehrere Alternativen für die Benutzerschnittstelle generiert werden sollen, sind die folgenden Parameter des Nutzungsprofils relevant:

- „Interactionflow.sequential.Preference“
- „Interactionflow.parallel.Preference“
- „Interactionflow.mutable.Preference“
- „Interactionflow.efficient.Preference“

In Anhang C sind im Rahmen der Darstellung der Funktion „analysiereAbstrakteInteraktion()“ die Schwellenwerte angegeben, ab denen der Algorithmus nur eine Alternative auswählt oder unterschiedliche Alternativen in einem neuen AIN bereitstellt.

Die Resulate der Bewertungsfunktion sind durch die roten Ziffern in Abbildung 66 visualisiert. Weiterhin ist auch dargestellt, wie die Bewertungsfunktion, die nur BSF in Bezug auf ein Nutzungsprofil bewerten kann, genutzt wird, um Aussagen über Interaktionsmetaphern und AINs zu treffen. Dies erfordert eine Betrachtung des Baumes von den Blättern bis zur Wurzel. Jedes Blatt des Baumes in Abbildung 66 ist ein BSF, da der Algorithmus so lange nach Varianten sucht, bis schließlich nur noch ein oder mehrere BSF für die Beschreibung verwendet werden können (vgl. Spezifikation des OM-Kataloges in Abschnitt 5.4 und Spezifikation der „Domain Theory“ in Abschnitt 7.3). Dabei ist anzumerken, dass der Baum in Abbildung 66 nur ein Ausschnitt des gesamten Baumes darstellt. Die fett hervorgehobenen, kurzen Pfeile geben an, dass sich der Baum hier fortsetzt. Die Ziffern an diesen kurzen Pfeilen geben den Wert der Bewertungsfunktion an, die aus den jeweiligen Kindern ermittelt wurde.

Für die Berechnung der Werte jedes einzelnen Knotens sind folgende Regeln zu beachten:

- Der Wert eines Knotens wird durch den höchsten Wert seiner Kinder bestimmt.
- Nur BSF sind Blätter im Baum. Deren Wert definiert die Bewertungsfunktion.
- Wird ein Knoten durch ein AIN beschrieben, so ermittelt sich sein Wert als Mittel aller Interaktionsmetaphern, die das AIN beschreiben. Ist ein BSF als Kind eines solchen Knoten angegeben, so wird der Wert des BSF gewählt, wenn dieser höher ist.
- Besitzt der Knoten ein Kind, das als „(Alt.-Komp)“ definiert ist, so ist der Wert dieses Knoten zu wählen.

Der vorletzte Punkt ist notwendig, da der in Abbildung 66 visualisierte Baum eine verkürzte Darstellung des Algorithmus nutzt. In einem Knoten vom Typ AIN wird sowohl eine Variante als auch das AIN selbst zusammengefasst. Da eine Variante sowohl über ein BSF als auch über ein AIN realisiert werden kann, müssen für die Bewertung auch die BSF hinzugezogen werden.

Die letzte Regel ist aufgrund einer Besonderheit des Algorithmus notwendig. Würde diese Regel nicht existieren, so könnte das durch die Komposition von Alternativen generierte AIN niemals gewählt werden, da dieses AIN (als Komposition aus bester und zweitbester Alternative) immer schlechter bewertet werden würde als die beste Alternative. Im Algorithmus (vgl. Funktion „analysiereAbstrakteInteraktion()“ in Anhang C) sind Schwellen angegeben, mit denen sichergestellt wird, dass ungeeignete Alternativen nicht für die Komposition ausgewählt werden können.

Legt man diese Definitionen zugrunde, so kann der Wert jedes Knotens eindeutig durch eine Bottom-up-Betrachtung ermittelt werden.

Die rot markierten Pfeile in Abbildung 66 heben die Pfade im Baum hervor, die zu den BSF führen, welche die Interaktionsmetapher „Rezept suchen“ bestmöglich realisieren. Betrachtet man diesen Pfad von der Wurzel ausgehend, so ist als erstes das AIN „SucheImKatalog_AIN“ ausgewählt worden.

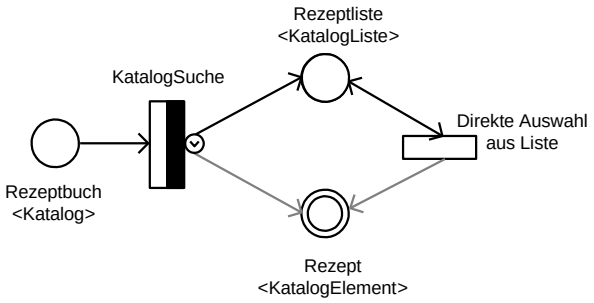


Abbildung 67: Ausschnitt aus dem OM-Katalog: SucheImKatalog_AIN

Wie in Abbildung 67 dargestellt, müssen nun Realisierungen für die beiden Interaktionsmetaphern „KatalogSuche“ und „Rezept auswählen“ gefunden werden. Die Interaktionsmetapher „KatalogSuche“ wurde, durch das generierte AIN „SucheInKategorien_StichwortSuche“ realisiert.

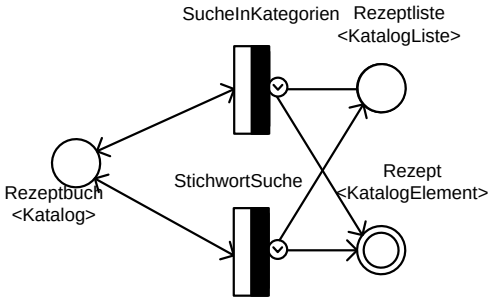


Abbildung 68: Realisierung der Interaktionsmetapher „KatalogSuche“ durch Kombination

Für die beiden in Abbildung 68 dargestellten Interaktionsmetaphern „SucheInKategorien“ und „StichwortSuche“ wurden, wie man dem Baum in Abbildung 66 entnehmen kann, die BSF „KatListe.bsf“ und „SuchForm.bsf“ ausgewählt.

Ersetzt man nun in der Interaktionsbeschreibung die Abstrakte Interaktion „Rezept suchen“ durch die ermittelten BSF, so erhält man das in Abbildung 69 dargestellt BSF-AIN. In rot sind die ermittelten BSF hervorgehoben. Für die weiteren Schritte der Komposition wurde aber die Beschriftung der Interaktionsbeschreibung beibehalten, um ein intuitiveres Verständnis der Bedeutung der einzelnen Stellen und Transitionen für die Benutzerschnittstelle zu ermöglichen.

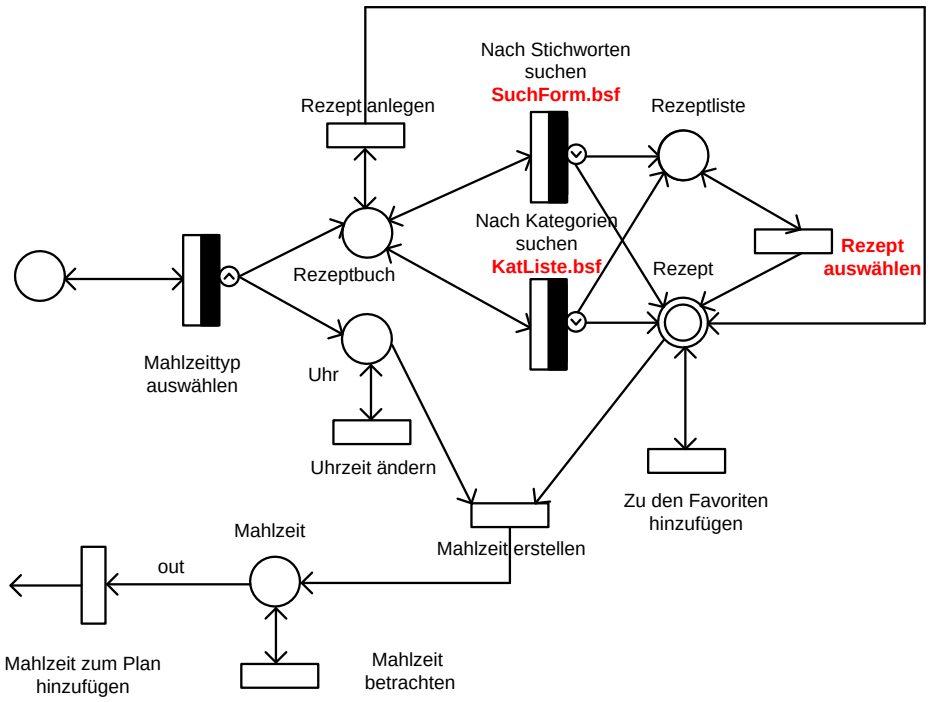


Abbildung 69: BSF-AIN am Beispiel „Youngster unterwegs mit dem PDA“

Mit dem BSF-AIN ist der erste Schritt des Generierungsprozesses, nämlich die Identifikation von passenden Benutzerschnittstellenfragmenten der „Domain Theory“ abgeschlossen.

Der nun folgende Schritt beschreibt die Komposition der identifizierten Benutzerschnittstellenfragmente basierend auf den Beziehungen, wie sie im BSF-AIN definiert sind.

9.1.4 Sequenzialisieren von parallelen Benutzerschnittstellenfragmenten

Im folgenden Schritt zur Generierung einer Benutzerschnittstelle erfolgt die ganzheitliche Betrachtung der im BSF-AIN ermittelten Benutzerschnittstellenfragmente. Während zur Identifikation der BSF Interaktionsmetaphern isoliert betrachtet wurden, muss nun eine Anpassung an das Zusammenspiel der BSF erfolgen, wie es im BSF-AIN definiert ist.

Ausgangslage dieser Betrachtung ist das Layoutpattern aus dem Nutzungsprofil „Youngster unterwegs mit dem PDA“. Abbildung 70 visualisiert dieses Layoutpattern, wie es in Abschnitt 8.4.1 eingeführt wurde.

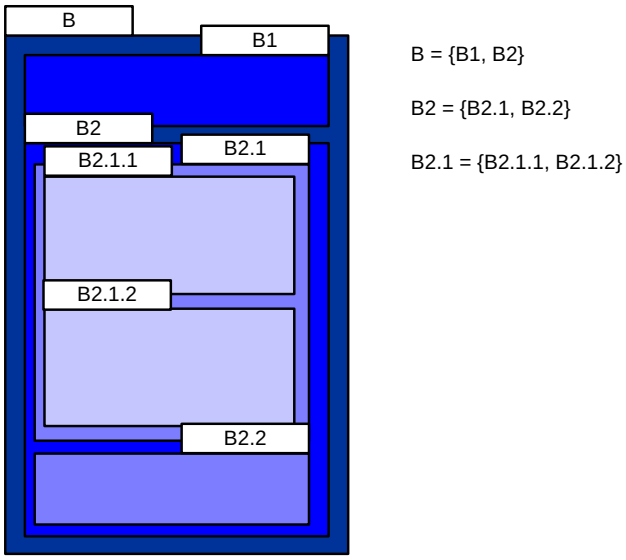


Abbildung 70: Layoutpattern „Youngster unterwegs mit dem PDA“

Basierend auf dem Layoutpattern kann jedem Benutzerschnittstellenfragment eine eindeutige Belegung von Bereichen zugeordnet werden, die während der Durchführung des BSF allokiert wird. Tabelle 17 veranschaulicht die Belegung der Benutzerschnittstellenfragmente aus dem gegebenen BSF-AIN auf das Layoutpattern.

Benutzerschnittstellenfragmente	B1	B2		
		B2.1		B2.2
		B2.1.1	B2.1.2	
„Rezept anlegen“				
„Nach Stichworten suchen“				
„Nach Kategorien suchen“				

„Rezept auswählen“				
„Uhrzeit ändern“				
„Mahlzeit erstellen“				
„Mahlzeit betrachten“				
„Mahlzeit hinzufügen“				
„Zu den Favoriten hinzufügen“				

Tabelle 17: BSF des BSF-AINs „Youngster unterwegs mit dem PDA“

Zusätzlich wurde in Tabelle 18 eine Belegung der Bereiche für die Präsentationsfragmente dargestellt, die durch die Stellen im BSF-AINs repräsentiert werden.

Präsentationsfragmente	B1	B2		
		B2.1		B2.2
		B2.1.1	B2.1.2	
„Rezeptbuch“				
„Rezeptliste“				
„Rezept“				
„Uhr“				
„Mahlzeit“				

Tabelle 18: PF des BSF-AIN „Youngster unterwegs mit dem PDA“

Zieht man nun das BSF-AIN aus Abbildung 69 heran, so wird offensichtlich, dass die dort modellierte parallele Durchführung von einigen dieser Interaktionen nicht auf dem Layoutpattern abbildbar ist. Um dieser Problemstellung gerecht zu werden, wurde ein Algorithmus (vgl. Abschnitt 8.4.4) entwickelt, der basierend auf einem „Brute-Force“-Ansatz das BSF-AIN modifiziert und anschließend überprüft, ob das Ergebnis der Modifikation den Anforderungen des ursprünglichen BSF-AIN gerecht wird. Für die automatisierte Evaluation der generierten BSF-AIN wurden entsprechende Methoden bereitgestellt, die basierend auf Pfad-Analysen im BSF-AIN arbeiten.

Die Methoden zur Modifikation des BSF-AIN wurden basierend auf den Möglichkeiten zur Sequenzialisierung von Abläufen im AIN erstellt.

Im Folgenden wird die Funktionsweise des Verfahrens anhand des BSF-AINs in Abbildung 69 und des hier vorgestellten Layoutpatterns (Abbildung 70) und den Tabellen für die Belegungen der BSF und PF (vgl. Tabelle 17, Tabelle 18) durchgeführt und evaluiert. Der in Abschnitt 8.4.4 präsentierte Algorithmus nutzt einen sogenannten Interaktionsbaum, um Konflikte im BSF-AIN zu identifizieren. Mit Hilfe des Interaktionsbaums lassen sich die folgenden Konflikte ermitteln:

1. Nachdem die Transition „Mahlzeit auswählen“ geschaltet hat und die Stellen „Uhr“ und „Rezeptbuch“ markiert sind, dürfen „Rezept anlegen“, „Nach Stichworten suchen“ und „Nach Kategorien

suchen“ aufgrund der Überschneidungen mit „Uhr“ und der jeweiligen Transition („Rezept anlegen“, „Nach Stichworten suchen“ und „Nach Kategorien suchen“) im Layoutpattern nicht mehr schalten.

2. Nachdem die Transition „Nach Stichworten suchen“ oder „Nach Kategorien suchen“ geschaltet hat und „Rezeptliste“ (Fall 1: „Rezept“) markiert wurde, können weder „Nach Stichworten suchen“ oder „Nach Kategorien suchen“ schalten, wenn „Rezept“ (Fall 2: „Rezeptliste“) markiert wurde.
3. Wenn die Stelle „Rezeptliste“ markiert ist, kann die Transition „Rezept anlegen“ nicht schalten.
4. Nachdem die Transition „Rezept auswählen“ ausgeführt worden ist, ist sowohl „Rezeptliste“ als auch „Rezept“ markiert. Dies ist aber nicht auf dem Layoutpattern darstellbar.
5. Wenn die Stelle „Mahlzeit“ markiert wurde, dürfen „Rezept Anlegen“, „Nach Stichworten Suchen“, „Nach Kategorien suchen“ und „Rezept auswählen“ nicht mehr schalten.
6. Die folgenden Transitionen können nicht gleichzeitig ausgeführt werden: „Rezept anlegen“, „Nach Sichtworten suchen“, „Nach Kategorien suchen“, „Rezept auswählen“, „Zu den Favoriten hinzufügen“, „Mahlzeit erstellen“ und „Mahlzeit betrachten“.

Im ersten Schritt wird der Algorithmus versuchen, Interaktionsabläufe zu überschreiben (vgl. **Methode 1** in Abschnitt 8.4.4). Hierfür wird das Schaltverhalten von Transitionen verändert, indem belegte Bereiche des Layoutpatterns für die Ausführung der Interaktion freigeben werden.

Transition	**Start**	**Ende**	**Aufgelöste Konflikte**
Rezept anlegen	{Rezept}- {Mahlzeit}- [Rezept auswählen]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten]-		Konflikt 3 Konflikt 5 Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6
Nach Stichworten suchen	{Rezeptliste}- {Rezept}- {Mahlzeit}- [Rezept auswählen]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten]-		Konflikt 2 Konflikt 2 Konflikt 5 Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6
Nach Kategorien suchen	{Rezeptliste}- {Rezept}- {Mahlzeit}- [Rezept auswählen]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten]-		Konflikt 2 Konflikt 2 Konflikt 5 Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6
Rezept auswählen	-	-	-

Uhrzeit ändern	-	-	-
Zu den Favoriten hinzufügen	-	-	-
Mahlzeit betrachten	-	-	-

Tabelle 19: Überschreiben von Interaktionsabläufen für „Youngster unterwegs mit dem PDA“

Tabelle 19 visualisiert die Änderungen im Schaltverhalten der Transitionen, die durch die Anwendung der Methode 1 (vgl Abschnitt 8.4.4) ermittelt werden. Dabei wurden die Transitionen in der Reihenfolge, wie sie in der Tabelle angegeben sind, durch die Methode 1 bearbeitet. Die Tabelle gibt das veränderte Schaltverhalten der Transitionen an, wobei in der zweiten Spalte („Start“) die Modifikationen angegeben sind, die durchgeführt werden, bevor die Transition schaltet und die Spalte „Ende“ die Modifikationen nach der Ausführung der Transition beschreibt. „-“ gibt an, dass eine Markierung aus einer Stelle abgezogen wird oder eine gestartete Transition beendet wird. „+“ beschreibt das Wiederherstellen von Markierungen oder gestarteten Transitionen. Stellen wurden durch geschweifte und Transitionen durch eckige Klammern gekennzeichnet. Die letzte Spalte beschreibt schließlich, welcher Konflikt durch die Modifikation behoben wird.

Wie man der Tabelle 19 entnehmen kann, wird an keiner Stelle eine Transition oder Stelle wieder hergestellt. Dies war im Rahmen des vorgestellten Layoutpatterns und der ermittelten BSF nicht zu erwarten, da der PDA nur über eine sehr begrenzte Präsentationsfläche verfügt, die von den benutzten BSF nahezu vollständig eingenommen wird. Weiterhin ist ersichtlich, dass nur für die ersten drei Transitionen Modifikationen angegeben wurden, während die folgenden Transitionen keine Modifikationen mehr aufweisen. Begründet ist dies dadurch, dass durch die bereits vorgenommenen Modifikationen der Interaktionsablauf bereits so stark sequenzialisiert wurde, dass keine weiteren Konflikte mehr auftreten können.

Am Beispiel der ersten Transition „Rezept anlegen“ werden einige Besonderheiten der vorgenommenen Modifikationen diskutiert. Im ersten Schritt muss festgehalten werden, dass Methode 1 nicht einfach alle Stellen und Transitionen entfernt, welche überschneidende Bereiche im Layoutpattern belegen. Methode 1 sieht vor, nur Modifikationen durchzuführen, die tatsächlich auch Konflikte auflösen. Ein Beispiel für einen Konflikt, der mit der Methode 1 nicht aufgelöst werden kann, ist der Konflikt zwischen der Transition „Rezept anlegen“ und der Stelle „Uhr“ und auch der Transition „Uhrzeit ändern“. Würde eine Markierung aus der Stelle „Uhr“ abgezogen werden, so würde das BSF-AIN nie mehr die Endtransition erreichen, da „Mahlzeit erstellen“ niemals schalten könnte. Auch wenn die Stelle „Uhr“ nach der Ausführung von „Rezept anlegen“ wieder hergestellt wird, wäre eine Markierung erreicht, die sich nicht auf das Layoutpattern abbilden lässt, da „Uhr“ und „Rezeptliste“ nicht gleichzeitig dargestellt werden können. Von den Alternativen „Rezept anlegen“, „Nach Stichworten suchen“ und „Nach Kategorien“ suchen, kann auch jeweils nur eine aktiv sein. Da diese Konflikte mit der Methode 1 nicht aufgelösst werden können, werden hier auch keine Änderungen am Schaltverhalten der Transitionen vorgenommen.

Fasst man die Modifikationen wie sie in Tabelle 19 dargestellt sind zusammen, so wird dadurch erreicht, dass „Rezeptliste“ und „Rezept“ niemals gleichzeitig markiert werden können.

Weiterhin werden durch die drei Transitionen „Rezept anlegen“, „Nach Stichworten Suchen“ und „Nach Kategorien Suchen“ gegebenenfalls aktive Transitionen beendet. Dies führt beispielsweise dazu, dass die Transition „Zu den Favoriten hinzufügen“ und die Transition „Mahlzeit betrachten“ nicht mehr parallel durchgeführt werden können (wie es noch im Ursprungsnetz möglich war). Damit beide gleichzeitig schalten können, muss sowohl „Rezept“, als auch „Mahlzeit“ markiert sein. Dieser Zustand lässt sich aber aufgrund des geänderten Schaltverhaltens nicht mehr erreichen.

Mit Hilfe der Methode 1 war es nicht möglich, die Konflikte 1 und 4 zu beheben. Daher wird der Algorithmus im nächsten Schritt **Methode 2** durchführen und versuchen, Nebenstellen aufzulösen (vgl. Methode 2 in Abschnitt 8.4.4). Betrachtet man das BSF-AIN, so weist es die folgenden Nebenstellen auf:

- {Rezeptbuch} <-> „Rezept anlegen“
- {Rezeptbuch} <-> „Nach Stichworten suchen“
- {Rezeptbuch} <-> „Nach Kategorien suchen“
- {Rezeptliste} <-> „Rezept auswählen“
- {Rezept} <-> „Zu den Favoriten hinzufügen“
- {Mahlzeit} <-> „Mahlzeit betrachten“

Durch das Auflösen der Nebenstelle {Rezeptliste}<->[Rezept auswählen] kann der Konflikt 4 behoben werden.

Im nächsten Schritt wird der Algorithmus versuchen die Methode „Auflösen durch Alternativen“ anzuwenden (vgl. **Methode 4** in Abschnitt 8.4.4). Da nur die Transition „Mahlzeittyp auswählen“ mehr als eine Ausgangsstelle besitzt, wird ein BSF-AIN, wie es in Abbildung 71 dargestellt ist, generiert.

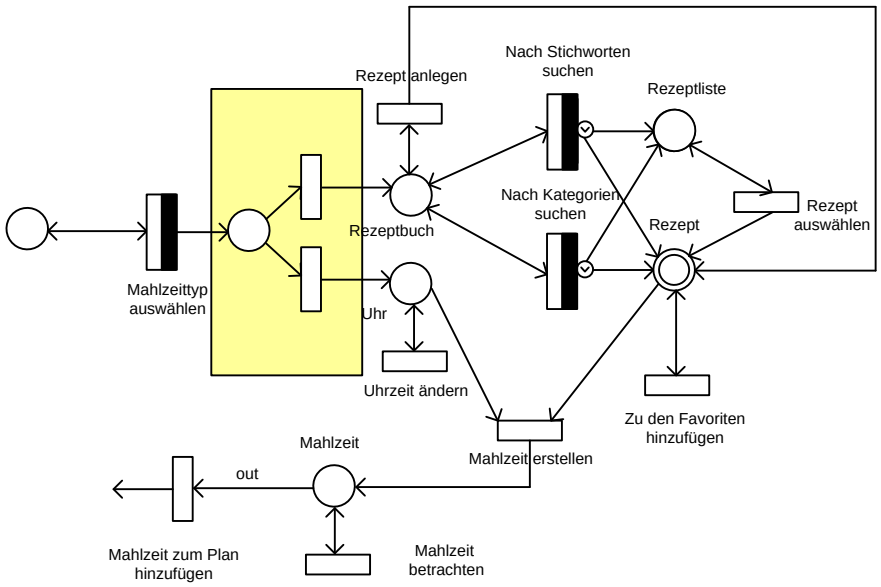


Abbildung 71: Anwendung der Methode „Auflösen durch Alternativen“ auf das BSF-AIN

Betrachtet man das modifizierte BSF-AIN aus Abbildung 71, so ist offensichtlich, dass die Transition „Mahlzeit erstellen“ niemals schalten wird, da nur eine der beiden neu generierten Transitionen schalten kann. Der Algorithmus würde dies erkennen und diese Methode verwerfen und mit dem nächsten Schritt fortfahren.

Im nächsten Schritt des Algorithmus wird die Methode „Auflösen durch Sequenzialisieren“ (vgl. **Methode 5** in Abschnitt 8.4.4) angewendet. Hierzu werden alle Stellen mit mehr als einer Ausgangsstelle (im gegebenen BSF-AIN ist das nur „Mahlzeit auswählen“) und alle Stellen mit mehr als einer Eingangstelle (im gegebenen BSF-AIN ist das nur „Mahlzeit erstellen“) betrachtet. Für das gegebene BSF-AIN werden die vier folgenden Varianten automatisiert generiert:

1. Kante **„Mahlzeittyp auswählen“ –> „Rezeptbuch“** wird aufgelöst;
   Kante **„Mahlzeit erstellen“ <- „Rezept“** wird aufgelöst;
   Neue Transition **„Rezept“ -> “<Transition>“ -> „Rezeptbuch“** wird erstellt;
2. Kante **„Mahlzeittyp auswählen“ –> „Rezeptbuch“** wird aufgelöst;
   Kante **„Mahlzeit erstellen“ <- „Uhr“** wird aufgelöst;
   Neue Transition **„Uhr“ -> “<Transition>“ -> „Rezeptbuch“** wird erstellt;
3. Kante **„Mahlzeittyp auswählen“ –> „Uhr“** wird aufgelöst;
   Kante **„Mahlzeit erstellen“ <- „Rezept“** wird aufgelöst;
   Neue Transition **„Rezept“ -> “<Transition>“ -> „Uhr“** wird erstellt;
4. Kante **„Mahlzeittyp auswählen“ –> „Uhr“** wird aufgelöst;
   Kante **„Mahlzeit erstellen“ <- „Uhr“** wird aufgelöst;
   Neue Transition **„Uhr“ -> “<Transition>“ -> „Uhr“** wird erstellt;

Variante 1 und Variante 4 führen zu Netzen, die über Stellen und Transitionen verfügen, die nicht mehr erreichbar sind. Der Algorithmus würde dies erkennen und diese Varianten verwerfen.

Variante 2 und Variante 3 führen zu Netzen, welche mögliche Ergebnisse liefern und den identifizierten Konflikt auflösen. Aus Sicht des Algorithmus sind beide Lösungen gleichwertig. Erst der Benutzerschnittstellen-Designer kann entscheiden, welche Lösung zu präferieren ist. Dies bedeutet aber auch, dass für beide Varianten Benutzerschnittstellen generiert werden müssen. Im Folgenden wird nur die Variante 3 weiter betrachtet. Abbildung 72 visualisiert das BSF-AIN nach der Durchführung der Sequenzialisierung. Zusätzlich ist das geänderte Schaltverhalten, wie es in Tabelle 19 dargestellt wurde, zu berücksichtigen.

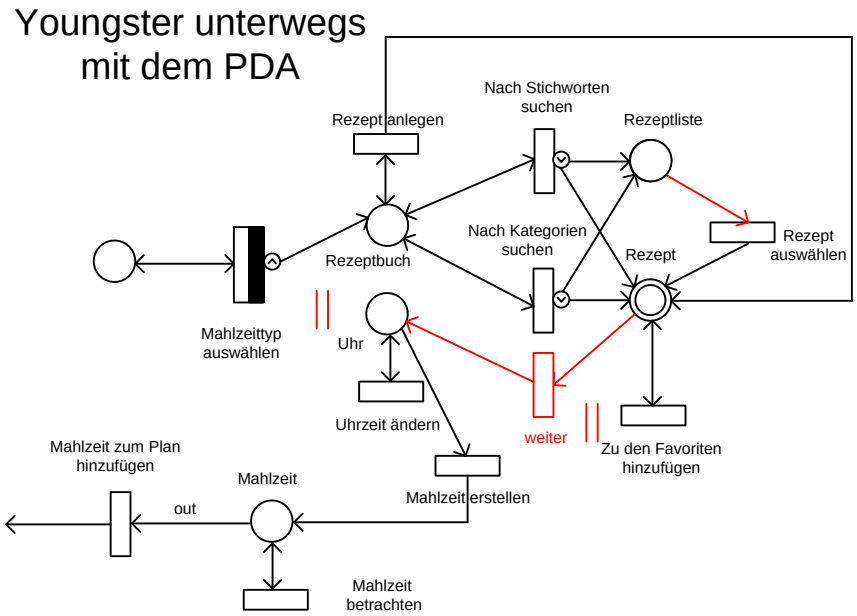


Abbildung 72: BSF-AIN „Youngster unterwegs mit dem PDA“ nach der Sequenzialisierung

9.1.5 Komposition basierend auf dem BSF-AIN

Ein BSF-AIN, das auf das gegebene Layoutpattern abgebildet werden kann, definiert schließlich die Datenbasis, auf der eine Komposition durchgeführt werden kann. Wie die Komposition der BSF durchzuführen ist, wurde in Abschnitt 8.4.5 erläutert. Eingesetzt wird dazu eine Intervallalgebra, die Beziehungen zwischen Benutzerschnittstellenfragmenten beschreibt, wie sie im BSF-AIN modelliert sind. Dabei wird die Durchführung eines Benutzerschnittstellenfragmentes als „Intervall“ aufgefasst. Jede Möglichkeit, wie sich zwei Intervalle überlappen können, definiert eine Relation der Intervallalgebra. Jeder dieser Relationen wurde ein Konstrukt aus dem BSF-AIN zugeordnet, so dass es möglich ist, aus dem BSF-AIN eine Liste von Relationen zu ermitteln. Für das in Abbildung 72 angegebene BSF-AIN lassen sich die folgenden Relationen identifizieren:

ALTERNATIVEN:

Rezeptbuch = ALT ([Rezept anlegen], [Nach Stichworten suchen], [Nach Kategorien suchen])

Rezept = ALT ([Zu den Favoriten hinzufügen], [weiter])

Uhr = ALT ([Uhrzeit ändern], [Mahlzeit erstellen])

Mahlzeit = ALT ([Mahlzeit betrachten], [Mahlzeit zum Plan hinzufügen]

NETZ:

[Mahlzeittyp auswählen] OVERLAPS Rezeptbuch

[Rezept anlegen] OVERLAPS Rezept

[Nach Stichworten suchen] OVERLAPS ([Rezept auswählen] OR Rezept])

[Nach Kategorien suchen] OVERLAPS ([Rezept auswählen] OR Rezept])

[Rezept auswählen] MEETS Rezept

[Weiter] MEETS Uhr

[Mahlzeit erstellen] MEETs Mahlzeit

NEBENSTELLEN:

[Zu den Favoriten hinzufügen] MEETS [Zu den Favoriten hinzufügen]

[Uhrzeit ändern] MEETS [Uhrzeit ändern]

[Mahlzeit betrachten] MEETS [Mahlzeit betrachten]

Die verwendete Synatx wurde in Abschnitt 8.4.5.10 eingeführt.

Die Realisierung der Komposition der Benutzerschnittstellenfragmente erfordert schließlich noch die Ausführung der einzelnen Relationen, wie sie in Abschnitt 8.4.5. spezifiziert wurden.

Ausgeführt werden kann die daraus generierte Spezifikation, wie in Abschnitt 7.2.2 beschrieben. Notwendig ist hierzu eine Laufzeitumgebung, die einerseits in der Lage ist, die definierte „Event-basierte"-Spezifikation auszuführen und andererseits die in den Benutzerschnittstellenfragmenten definierten XML-basierten Benutzerschnittstellenspezifikationen interpretieren kann. Im folgenden Abschnitt wird anhand der generierten Benutzerschnittstelle diskutiert, wie sich die Komposition auf die Benutzerschnittstelle ausgewirkt hat.

9.1.6 Diskussion der generierten Benutzerschnittstellen

Im Folgenden werden Ausschnitte aus dem Interaktionsablauf anhand der generierten Benutzerschnittstellen präsentiert. Dabei wird basierend auf dem BSF-AIN und den durchgeführten Kompositionsanweisungen die Umsetzung der dort gegebenen Spezifikation diskutiert.

Abbildung 73 zeigt einen Screenshot der Benutzerschnittstelle des iPhones, nachdem der Benutzer die Interaktion „Mahlzeittyp auswählen" ausgeführt hat. Rechts in Abbildung 73 ist zur Erläuterung das BSF-AIN dargestellt, das zusätzlich durch eine „rote Markierung" ergänzt wurde. Die Markierung gibt an, welche BSF aktiviert sind, die zu dem dargestellten Screenshot geführt haben.

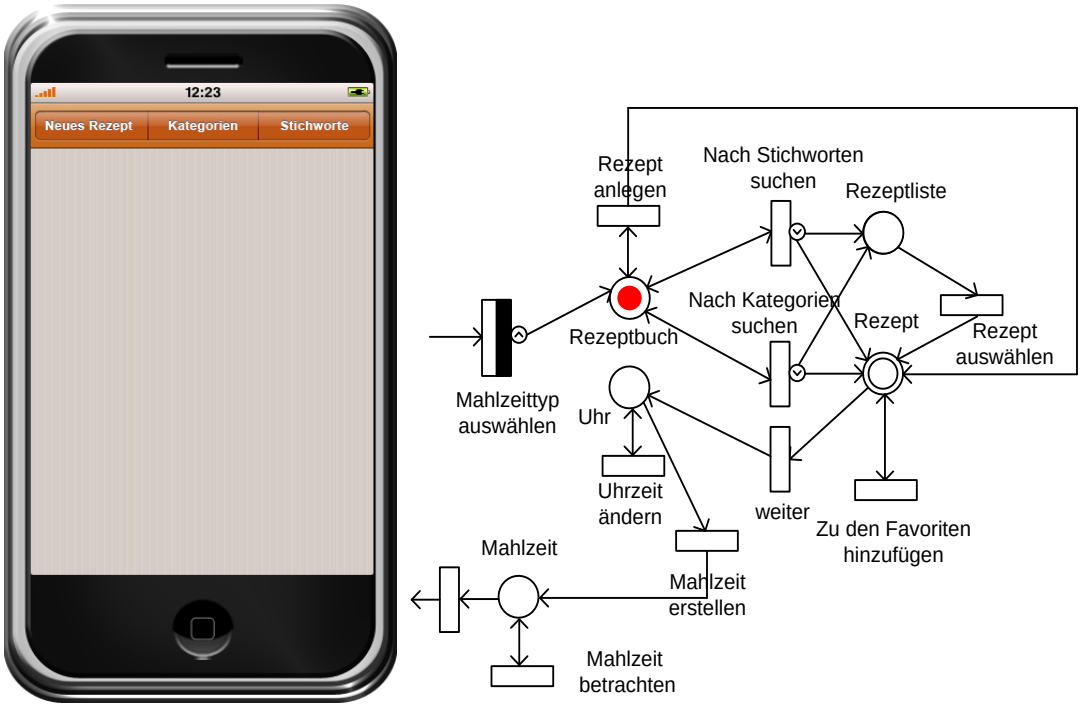


Abbildung 73: Screenshot: „Youngster unterwegs mit dem PDA“-„Rezeptbuch“

Wie man Abbildung 73 entnehmen kann, ist nur die Stelle „Rezeptbuch“ aktiviert. Stellen im BSF-AIN sind direkt auf Präsentationsfragmente abbildbar. Bei dem Präsentationsfragment, welches „Rezeptbuch“ realisiert, handelt es sich um eine Variante eines Menüs, das aufgrund des Layoutpatterns auf dem oberen Bereich abgebildet wird.

Die Stelle „Rezeptbuch“ wurde durch die Alternativenrelation erweitert. Die drei BSF „Rezept anlegen“, „Nach Stichworten suchen“ und „Nach Kategorien suchen“, wurden über diese Relation miteinander komponiert. Die drei BSF teilen sich eine gemeinsame Eingangsstelle. Für alle drei BSF wurde ihre „Starts“-Methode ausgeführt. Das bedeutet, dass alle drei BSF initialisiert sind und ausgeführt werden können. Die Alternativenrelation stellt weiterhin sicher, dass nur eine der Alternativen gleichzeitig aktiv sein kann.

Die Komposition über die Alternativenrelation bedeutet für das Präsentationsfragment, dass es durch Verschmelzung der Eingangsstellen der jeweiligen BSF erweitert wird. In dem in Abbildung 73 dargestellten Fall sind das nur Interaktionsobjekte („Buttons“).

Die Ausgestaltung des Präsentationsfragmentes, also die farbliche Darstellung, Umbenennung der Buttons, Reihenfolge und Positionierung der Interaktionsobjekte, ist Aufgabe des „Benutzerschnittstellen-Designers“, und nicht mehr Teil des automatisierten Generierungsprozesses.

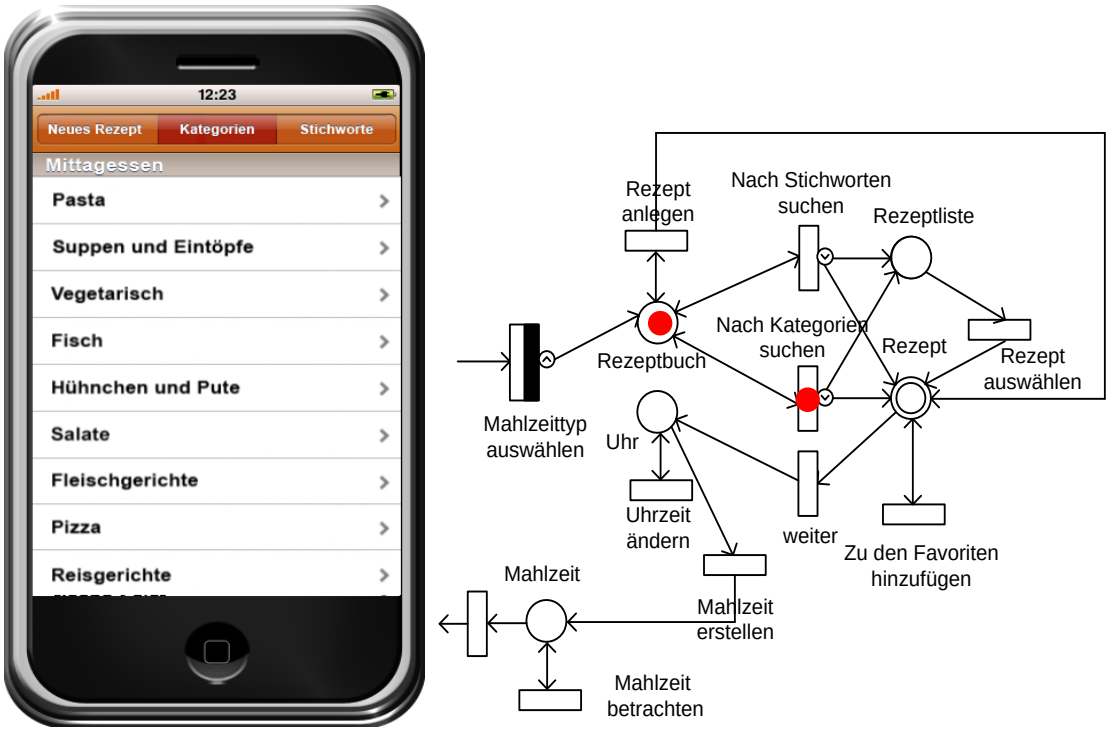


Abbildung 74: Screenshot: „Youngster unterwegs mit dem PDA“-„Nach Kategorien suchen“

Abbildung 74 visualisiert die Ausführung des BSF „Nach Kategorien suchen“ (eine der drei Alternativen). Dargestellt ist in Abbildung 74 nur ein Ausschnitt in der Ausführung des BSF, das die Navigation in Kategorien ermöglicht. Für die Realisierung der Kategoriesuche wurde eine iPhone spezifische Umsetzung ausgewählt, die das horizontale Blättern in Kategorien und deren Subkategorien ermöglicht.

BSF können beliebig viele Präsentationsfragmente und komplexe Interaktionsabläufe umfassen. Im BSF-AIN wird daher dem Schalten einer Transition ein ganzer Zeitraum zugewiesen. Sind in diesem Zeitraum neben den Präsentationsfragmenten des aktuell schaltenden BSF noch andere Präsentationsfragmente sichtbar, so können durch diese neue BSF gestartet werden, die zum bereits Laufenden parallel ausgeführt werden oder sich unterbrechen können. Die BSF „Nach Kategorien suchen“ „Rezept anlegen“ und „Nach Stichworten suchen“ sind über die Alternativenrelation verknüpft, so dass hier keine Parallelität auftreten kann.

Das BSF „Nach Kategorien suchen“ ist beendet, wenn ein Blatt des Kategorienbaums erreicht wird. Wie durch das „v“ Symbol an der Transition „Nach Kategorien suchen“ dargestellt, handelt es sich um eine „benutzerinitiierte Systeminteraktion“ (vgl. Abschnitt 5.3.3). Das bedeutet, dass erst zur Laufzeit (abhängig von der Anzahl der Rezepte der ausgewählten Kategorie) ermittelt wird, ob die Ausgangsstelle der Transition „Rezeptliste“ (mehr als ein Rezept in der Kategorie) oder „Rezept“ (keins oder genau ein Rezept) ist.

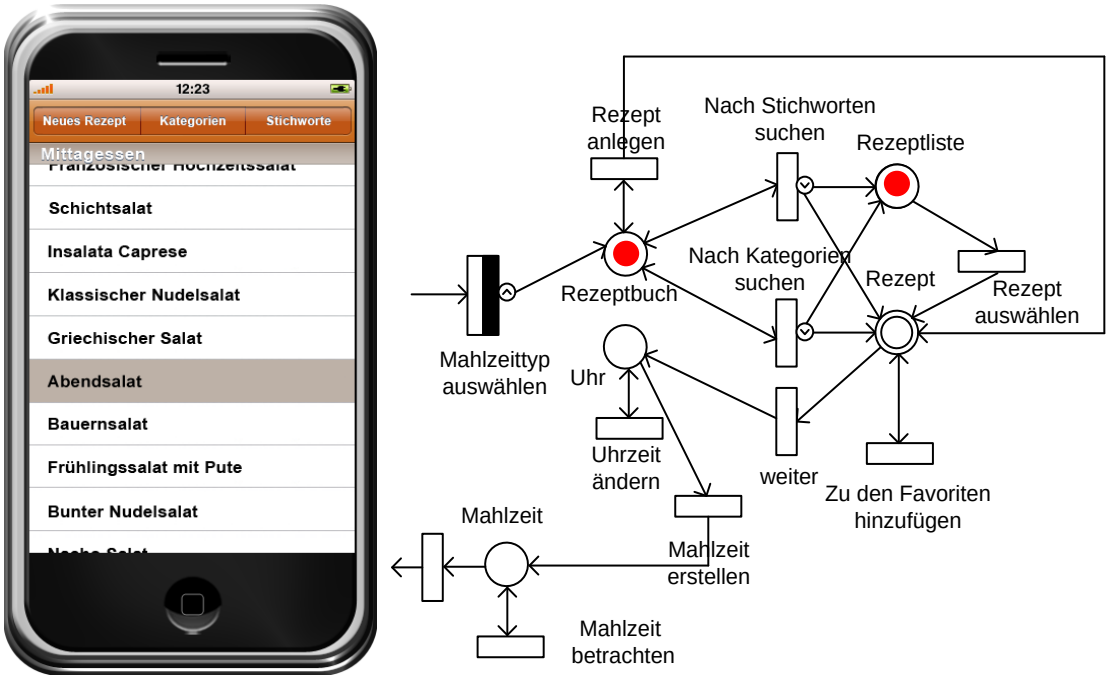


Abbildung 75: Screenshot: „Youngster unterwegs mit dem PDA“-„Rezeptbuch“ & „Rezeptliste“

Abbildung 75 zeigt einen Screenshot, nachdem das BSF „Nach Kategorien suchen“ abgeschlossen wurde. Wie man dem dazugehörigen BSF-AIN entnehmen kann, sind die Stellen „Rezeptbuch“ und „Rezeptliste“ markiert. Für die Realisierung des Präsentationsfragmentes „Rezeptliste“ wurde vom Algorithmus eine scrollbare Liste (iPhone spezifisch) ausgewählt, welche die Rezepte anzeigt, die der ausgewählten Kategorie zugeordnet sind.

Blickt man auf die Komposition zurück (vgl. Abschnitt 9.1.5), so wurde das BSF „Nach Kategorien suchen“ über die Overlaps-Relation mit dem BSF „Rezept auswählen“ verknüpft. Diese Verknüpfung wird realisiert, indem das Enddialogfragment der Relation „Nach Kategorien suchen“ einerseits das „Startdialogfragment“ des BSF „Rezept auswählen“ aufruft und damit dieses BSF aktiviert. Anderseits wird auch das eigene „Startdialogfragment“ aufgerufen, so dass die Transition „Nach Kategorien suchen“ selbst wieder aktiviert wird. Da auch das BSF „Nach Stichworten suchen“ über die Alternativenrelation mit „Rezept anlegen“ und „Nach Kategorien suchen“ verknüpft ist, teilen sich diese Transitionen ein „Startdialogfragment“, so dass auch die anderen beiden Alternativen wieder verfügbar werden.

Erwähnenswert ist weiterhin eine Besonderheit des BSF „Rezept auswählen“. Dieses BSF wird als „trivial“ bezeichnet, da es nur aus einem Präsentationsfragment („Rezeptliste“) besteht, und nur durch ein Interaktionsobjekt (im Fall „Rezeptliste“ nur „Auswahl eines Buttons“), das Teil dieses Präsentationsfragmentes ist, realisiert wird. Die Ausführung solcher trivialer BSF erfolgt wie bei allen anderen BSF. Auch die Ausführung von trivialen BSF kann einen Zeitraum in Anspruch nehmen, da Interaktionsobjekte in diesem Zeitraum selbst komplex sein können. Ist beispielsweise das Interaktionsobjekt eine interaktive Landkarte, so kann auch das triviale BSF „Ort suchen“ komplex sein.

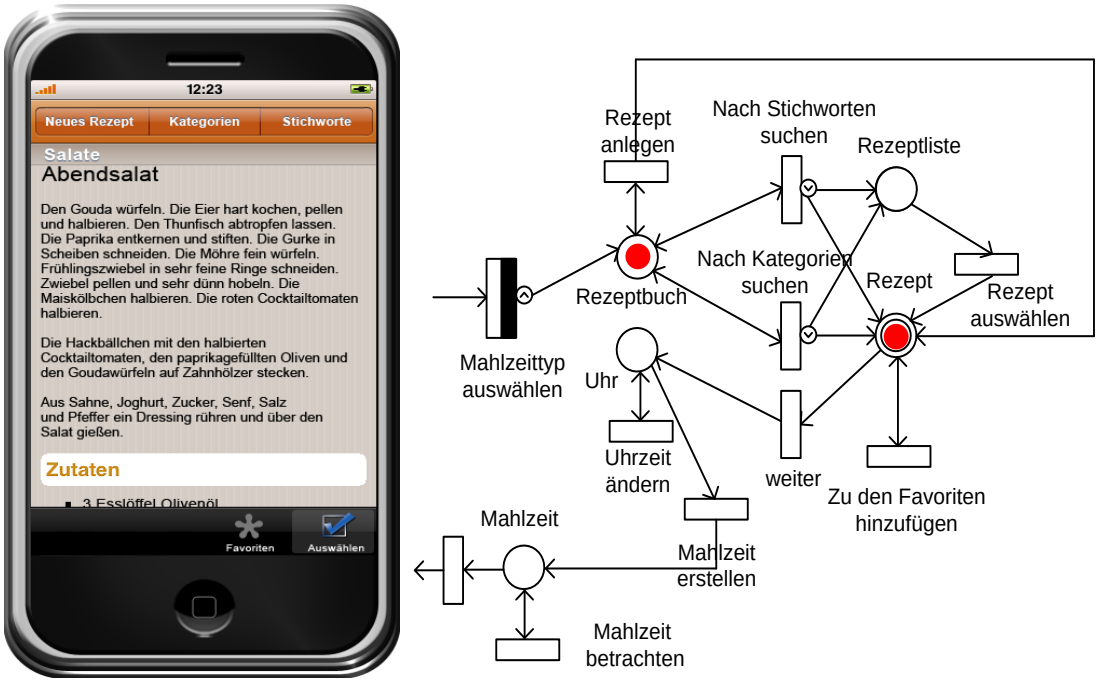


Abbildung 76: Screenshot: „Youngster unterwegs mit dem PDA“-„Rezept“

Abbildung 76 zeigt einen Screenshot nach der Auswahl eines Rezeptes. Dargestellt wird die Realisierung des Präsentationsfragmentes Rezept als scrollbares „Content“-Objekt, das die Kochanleitung, eine Zutatenliste und auch eine Ernährungsampel umfasst. Dieses Objekt wurde nicht generiert, sondern als Benutzerschnittstellenfragment der „Domain Theory“ hinzugefügt. Die Konzeption und Erstellung solcher Content-Objekte ist höchst komplex und kann nur schwer automatisiert werden. Die Möglichkeit wieder verwendbar Zugriff auf ein solches Objekt zu haben und dieses für die Erstellung neuer Anwendungen nutzen zu können, ist eine der wesentlichen Stärken des hier verfolgten Ansatzes.

Die Relation, die für die Komposition verwendet wurde, lautet: [Rezept auswählen] MEETS [ALT[Zu den Favoriten hinzufügen], [weiter]). Verwendet wurde die Meets-Relation, die sicherstellt, dass das aktuelle BSF (im Beispiel „Rezept auswählen“) beendet ist, und das Nachfolgende gestartet wird. Dabei kann der Nachfolger sowohl das BSF „Zu den Favoriten hinfügen“ als auch „weiter“ sein. Da beide über die Alternativenrelation miteinander verknüpft sind, verfügen sie über ein gemeinsames Startdialogfragment, das nach dem Ausführen von „Rezept wählen“ aktiviert wird.

An dieser Stelle sei darauf hingewiesen, dass im ursprünglichen AIN in der Interaktionsbeschreibung an dieser Stelle noch die Overlaps-Relation verwendet wurde. Durch die Sequenzialisierung wurde diese in die Meets-Relation umgewandelt, da „Rezept“ und „Rezeptliste“ den gleichen Bereich im Layoutpattern einnehmen und somit nicht gleichzeitig ausgeführt werden können.

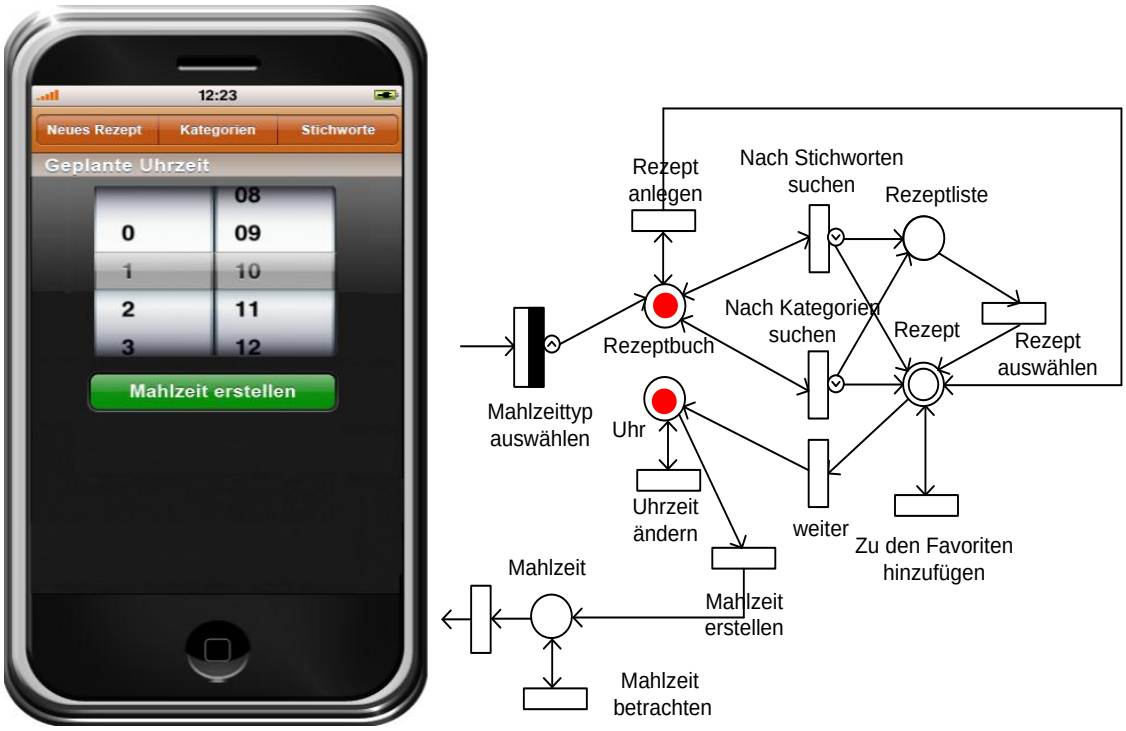


Abbildung 77: Screenshot: „Youngster unterwegs mit dem PDA“-„Mahlzeit erstellen“

Abbildung 77 zeigt einen Screenshot nachdem „weiter“ ausgeführt wurde. Sichtbar ist das Präsentationsfragment „Uhr“, wobei laut BSF-AIN die Interaktionen „Uhrzeit ändern“ und „Mahlzeit erstellen“ verfügbar sind. Wie man dem Screenshot entnehmen kann, ist das BSF „Mahlzeit erstellen“ über einen Button erreichbar. „Uhrzeit ändern“ wird realisiert, indem der Benutzer über das Touch-Display des iPhones die Stunden und Minuten in Form eines „Wahlrades“ manipulieren kann. Im BSF-AIN ist „Uhr ändern“ als Transition modelliert, die nur über eine Nebenstelle (im Beispiel „Uhr“) verfügt. Komponiert werden solche BSF über eine Meets-Relation mit sich selbst. Das bedeutet für den Interaktionsablauf aber, dass durch diese Interaktion keine neue Interaktion ausgelöst wird, sondern dass das Präsentationsfragment weiterhin (wenn auch modifiziert) verfügbar bleibt.

Das BSF-AIN zu dem Screenshot veranschaulicht eine weitere Besonderheit der Komposition. Wie aus dem BSF-AIN (und dem Screenshot) ersichtlich ist, ist neben dem Präsentationsfragment „Uhr“ auch noch „Rezeptbuch“ sichtbar und für den Nutzer erreichbar. Dies bedeutet aber auch, dass der Benutzer jederzeit die Möglichkeit hat, andere Interaktionen (im Beispiel: „Rezept anlegen“, „Nach Stichworten suchen“ oder „Nach Kategorien suchen“) zu starten. Die in Abschnitt 9.1.4 ermittelten Erweiterungen der jeweiligen BSF zum „Überschreiben von Interaktionsabläufen“ stellen nun sicher, dass gerade ausgeführte BSF unterbrochen bzw. Präsentationsfragmente beendet werden. Für das Beispiel würde es bedeuten, dass das Präsentationsfragment „Uhr“ beendet wird und damit den belegten Platz für die Ausführung der gewählten Interaktion freigibt.

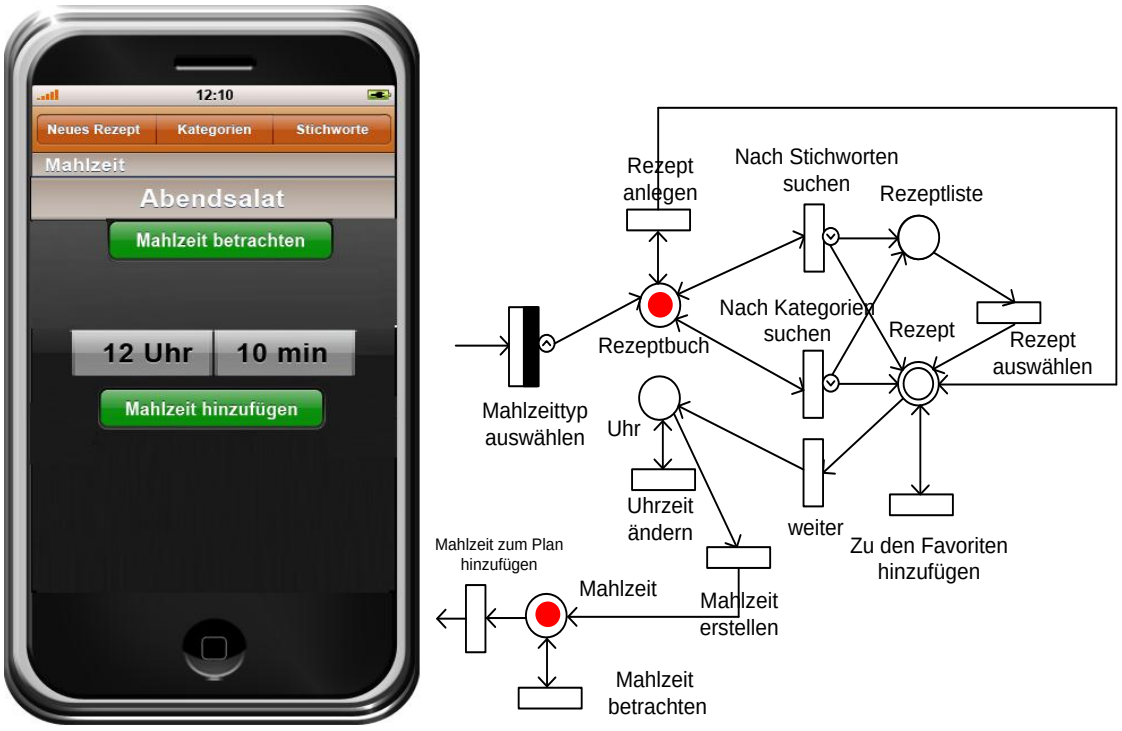


Abbildung 78: Screenshot: „Youngster unterwegs mit dem PDA“-„Mahlzeit“

Abbildung 78 visualisiert schließlich den letzen Schritt des Ablaufes zur Erstellung einer Mahlzeit.

Durch das BSF-„Mahlzeit zum Plan hinzufügen“ wird die Ausgangsstelle des Netzes erreicht. Das Netz ist damit abgeschlossen und alle Präsentationsfragmente und alle noch aktiven BSF werden beendet.

9.2 „Mid Ager bei der Arbeit am PC“

Im zweiten Schritt der Evaluation wird der Generierungsprozess einer Benutzerschnittstelle für die Interaktionsbeschreibung „Mahlzeit anlegen“ und den Nutzungskontext „Mid Ager bei der Arbeit am PC“ beschrieben. Der Generierungsprozess nutzt somit dieselbe Interaktionsbeschreibung und dieselbe Wissensbasis bestehend aus OM-Katalog, Nutzungskontextmodell und „Domain Theory“, wie sie auch in dem vorhergegangenen Abschnitt verwendet wurde. Lediglich der Nutzungskontext, der als Eingabe für die Generierung verwendet wird, unterscheidet sich.

Dieser zweite Schritt dient dabei nicht nur als zweites Evaluationsbeispiel, sondern soll zeigen, dass wenn die benötigte Wissensbasis zur Verfügung steht, eine weitgehend automatisierte Generierung von Benutzerschnittstellen nur durch die Definition von neuen Nutzungskontexten möglich wird.

Für die Präsentation des Generierungsprozesses wurde dieselbe Kapitelstruktur (wie beim Nutzungskontext „Youngster unterwegs mit dem PDA“) gewählt, um Unterschiede in den einzelnen Schritten des Generierungsprozesses aufzudecken und darzustellen, wie sich ein geänderter Nutzungskontext auf die einzelnen Methoden auswirkt.

9.2.1 Modellierung der Interaktionsbeschreibung

Die Interaktionsbeschreibung spezifiziert die Interaktion des Benutzers, ohne Aussagen über das verwendete Endgerät, den Nutzer und die Umgebung der Nutzung zu treffen. Sie spezifiziert somit auf einer abstrakten Ebene die zu generierende Benutzerschnittstelle und ist somit für alle Nutzungskontexte gleich.

Der im folgenden dargestellte Generierungsprozess nutzt die Interaktionsbeschreibung, wie sie in Abschnitt 9.1.1 vorgestellt wurde, und damit auch alle Erweiterungen des OM-Kataloges, die notwendig für die Erstellung der Interaktionsbeschreibung waren.

Im Rahmen des Prozessschrittes „Evaluation und Modifikation“ (vgl. Abschnitt 8.1), welcher vom Benutzerschnittstellen-Designer vorgenommen wird, können aber Erweiterungen des OM-Kataloges oder der „Domain Theory“ vorgenommen werden, die sich auf den Generierungsprozess auswirken. Um das Verfahren besser bewertbar zu machen, wird im Rahmen der Evaluation auf solche Änderungen verzichtet. Änderungen der Wissensbasis wirken sich auf alle Benutzerschnittstellen aus, wenn die Generierung für den selben Nutzungskontext wiederholt durchgeführt wird.

9.2.2 Modellierung des Nutzungskontextes und Generierung des Nutzungsprofils

Der Nutzungskontext definiert die Anforderungen an die zu generierende Benutzerschnittstelle. Somit bestimmen die Unterschiede zwischen dem Nutzungskontext „Youngster unterwegs mit dem PDA“, für den in den vorhergehenden Abschnitten eine Benutzerschnittstelle generiert wurde, und dem in Tabelle 20 präsentierten Nutzungskontext „Mid Ager bei der Arbeit am PC“, die Faktoren, die für die Generierung unterschiedlicher Benutzerschnittstellen verantwortlich sind.

	Umgebung	Nutzer	Endgerät
Default-Profil	Arbeit	Nutzer	PC
Kontext-Element	- Lichtverhältnisse: ++ - Personen im Umfeld: - - Ablenkung: +	- Sehvermögen: + - Motorik: +	

Tabelle 20: Nutzungskontext-„Mid Ager bei der Arbeit am PC“

Für die Generierung der Benutzerschnittstelle ist eine Spezifikation eines Nutzungskontextes, wie er in Tabelle 20 präsentiert wird, ausreichend. Das Nutzungskontextmodell, welches dieser Spezifikation zugrunde liegt und Defaultprofile und Kontextelemente definiert, bleibt für alle damit beschriebenen Nutzungskontexte unverändert.

In Abschnitt 6.4 wurde ein Nutzungskontextmodell für die Anwendungsdomäne des „Adipositas-Begleiter“ eingeführt. Dieses bildet die Grundlage für die Spezifikation der Nutzungskontexte und schließlich für die Generierung der Nutzungsprofile.

Auf die Beschreibung der einzelnen Kontextelemente des Nutzungskontextes „Mid Ager bei der Arbeit am PC“ wird verzichtet. Diese findet sich in Abschnitt 6.4 dieser Arbeit. Stattdessen werden die Unterschiede und Besonderheiten zu dem vorhergegangenen Nutzungskontext „Youngster unterwegs mit dem PDA“ anhand der drei Kontextkomponenten diskutiert:

- Kontextkomponente Umgebung
  Das Defaultprofil „Arbeit“ beschreibt die Nutzung einer Anwendung in einem beruflichen Umfeld am Arbeitsplatz. Es zeichnet sich dadurch aus, dass es im Gegensatz zum Defaultprofil „Unterwegs“ auch die Möglichkeit zum konzentrierten Umgang mit der Anwendung bietet. Gemeinsame Anforderungen liefern beide Profile im Bezug auf die Notwendigkeit, Unterbrechungen im Interaktionsablauf vorzusehen. Der Arbeitsplatz wird dabei (wie auch im Defaultprofil „Unterwegs“) nicht als ein privates Umfeld angesehen, so dass Personen im Umfeld oder auch andere Ablenkungen berücksichtigt werden müssen.
  Für die Benutzerschnittstelle bedeutet dies, dass im Gegensatz zum Defaultprofil „Unterwegs“ auch komplexe Interaktionsobjekte angeboten werden können, und sich die Benutzerschnittstelle auch dynamischer Interaktionsabläufe (beispielsweise parallele Darstellung von Teilen der Benutzerschnittstelle, die über unterschiedliche Fenster verteilt sind) bedienen kann. Multimediale Inhalte (insbesondere Audio Inhalte) sollten vermieden werden, da sich fremde Personen im Umfeld befinden können.
- Kontextkomponente Nutzer
  Das Defaultprofil „Nutzer“ beschreibt die neutralste der drei für den „Adipositas-Begleiter“ definierten Defaultprofile für diese Kontextkomponente. Es beschreibt eine Person, die sowohl mit dem Endgerät als auch mit den für dieses Endgerät typischen Benutzerschnittstellen vertraut ist und diese uneingeschränkt bedienen kann. Ein Unterschied im Vergleich zum Defaultprofil „Technikaffiner Nutzer“ liegt in der Präferenz von effizienten Interaktionsabläufen vor, die mit möglichst wenigen Interaktionsschritten zu einem Ergebnis führen.
  Für die Benutzerschnittstelle bedeutet dies, dass auch eine hohe Informationsdichte von Benutzerschnittstellen und die Parallelität von Interaktionsabläufen akzeptiert werden. Multimediale Inhalte wie Video und Audio sollten zugunsten von textuellen und graphischen Inhalten vermieden werden. Flexibilität, indem Alternativen im Interaktionsablauf angeboten werden und unterschiedliche Wege zur Zielerreichung existieren, wird nicht bevorzugt.

- Kontextkomponente Endgerät
  Das Endgerät PC bietet im Gegensatz zum mobilen Endgerät eine erheblich größere Präsentationsfläche und ermöglicht durch die gleichzeitige Nutzung von Maus und Tastatur höchste Flexibilität in der Bedienung von Interaktionsobjekten. Eine Besonderheit ist die Möglichkeit, dynamisch neue Fenster generieren zu können. Dieser Mechanismus ermöglicht es, die Präsentationsfläche nahezu beliebig zu erweitern.

Wie man dem Nutzungskontext „Mid Ager bei der Arbeit am PC“ entnehmen kann, liefert es eine Anforderungsliste, welche auch komplexere Interaktionsabläufe ermöglicht. Während der Nutzungskontext „Youngster unterwegs mit dem PDA“ durch die Verwendung des mobilen Endgerätes (kleines Display, Interaktionen nur über das Touch-Display) und des Defaultprofils „Unterwegs“ diesbezüglich noch eingeschränkt war, bietet „Mid Ager bei der Arbeit am PC“ weitgehende Flexibilität.

9.2.3 Generierung des BSF-AINs

Die Generierung des BSF-AINs nutzt die Interaktionsbeschreibung als Ausgangslage und ermittelt eine passende Abbildung auf die BSF der „Domain Theory“ mit Hilfe des OM-Kataloges und der Bewertungsfunktion, die sich des Nutzungsprofils und der „Domain Theory“ bedient.

Um die Vergleichbarkeit der Generierungsdurchläufe (für „Youngster unterwegs mit dem PDA“ und „Mid Ager bei der Arbeit am PC“) zu gewährleisten, erfolgt die Diskussion der Generierung des BSF-AINs anhand eines Baumes, wie er in Abschnitt 9.1.3 für „Youngster unterwegs mit dem PDA“ beschrieben wurde.

Abbildung 79 visualisiert die Arbeitsschritte des Algorithmus zur Generierung des BSF-AIN (vgl. Abschnitt 8.2.2) für die Interaktionsmetapher „Rezept suchen“ und das Nutzungsprofil „Mid Ager bei der Arbeit am PC“. Wie man der Abbildung 79 entnehmen kann, stimmt die Struktur des Baumes weitgehend mit dem Baum für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ überein. Die Ähnlichkeit ergibt sich aus der Verwendung desselben OM-Kataloges. Dieser gibt alle Möglichkeiten vor, wie die einzelnen Interaktionsmetaphern zu zerlegen sind. Im dargestellten Baum werden alle Varianten betrachtet, die durch den OM-Katalog beschrieben sind, so dass dieser für alle Nutzungsprofile ähnlich aufgebaut ist. Erst die Bewertungsfunktion ermöglicht eine Endscheidungsfindung, welche BSF für das gegebene Nutzungsprofil geeignet sind. Entsprechend unterscheiden sich die Werte der Knoten in dem in Abbildung 79 präsentierten Baum erheblich von dem Baum für den Nutzungskontext „Youngster unterwegs mit dem PDA“.

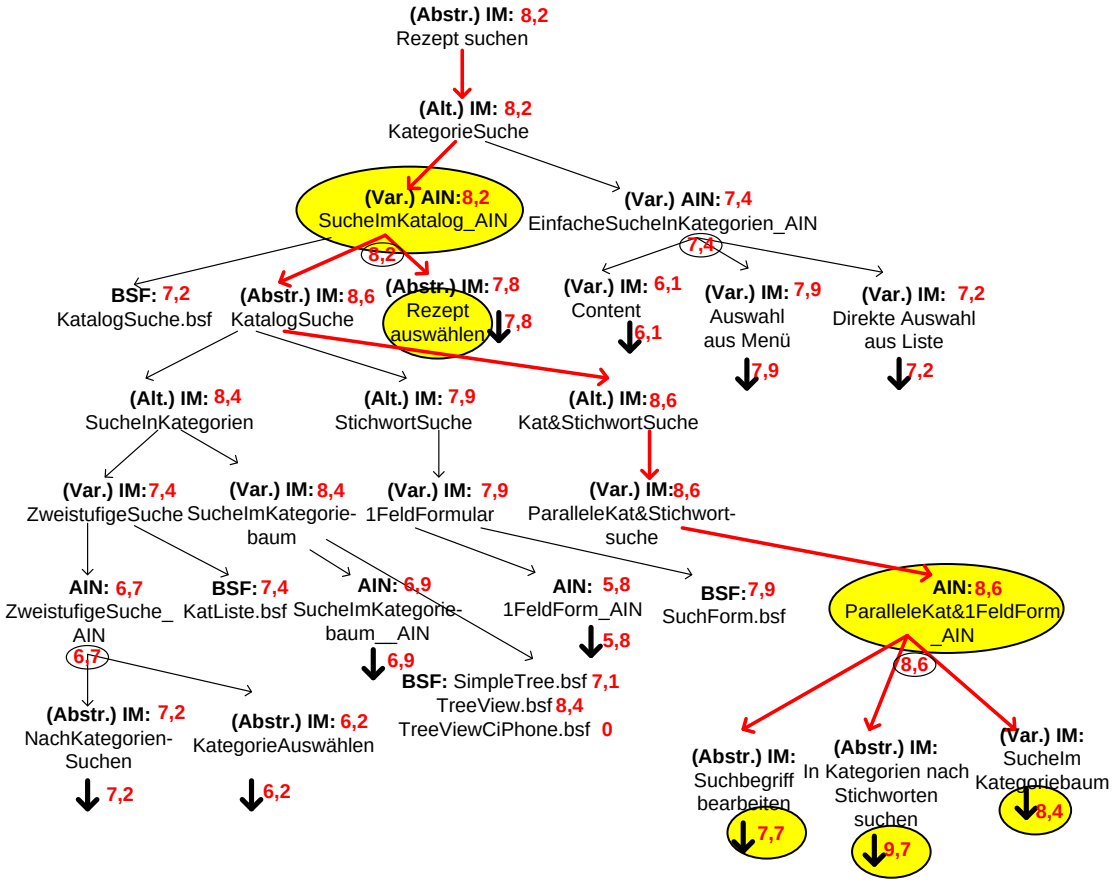


Abbildung 79: Ermittlung des BSF-AINs – „Mid Ager bei der Arbeit am PC“

Neben den Werten der Bewertungsfunktion findet sich ein weiterer Unterschied (zwischen den Bäumen der beiden Nutzungskontexte) in der Struktur des Baumes. Betrachtet man die Möglichkeiten zur Realisierung der Interaktionsmetapher „KatalogSuche“ so findet sich für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ neben der Betrachtung der einzelnen Alternativen auch noch die „Komposition von Alternativen“ als mögliche Lösung. Im Beispiel des Nutzungsprofils „Youngster unterwegs mit dem PDA“ lieferte sogar die Komposition von Alternativen die am besten bewertete Lösung. Aufgrund des Nutzungsprofils „Mid Ager bei der Arbeit am PC“, das spezifiziert, dass variantenfreie Interaktionsabläufe favorisiert werden, wurde die Komposition von Alternativen niedriger priorisiert, so dass sie für dieses Nutzungsprofil nicht in die Lösungsmenge aufgenommen wurden.

Vergleicht man die ermittelten BSF der beiden Nutzungsprofile, so wählen beide das AIN „SucheImKatalog_AIN“ als beste Lösung. Unterschiede finden sich aber bereits bei der Wahl der Alternativen für die Realisierung der Interaktion „KatalogSuche“. Für das Nutzungsprofil „Mid Ager bei der Arbeit am PC“ wurde hier die Alternative „Kat&StichwortSuche“ gewählt, welche durch das AIN „ParalleleKat&1FeldForm_AIN“ (vgl. Ausschnitt des OM-Kataloges in Tabelle 16) beschrieben wird. Legt man diese Auswahl zugrunde, so wird ein BSF-AIN generiert, wie es in Abbildung 80 dargestellt ist.

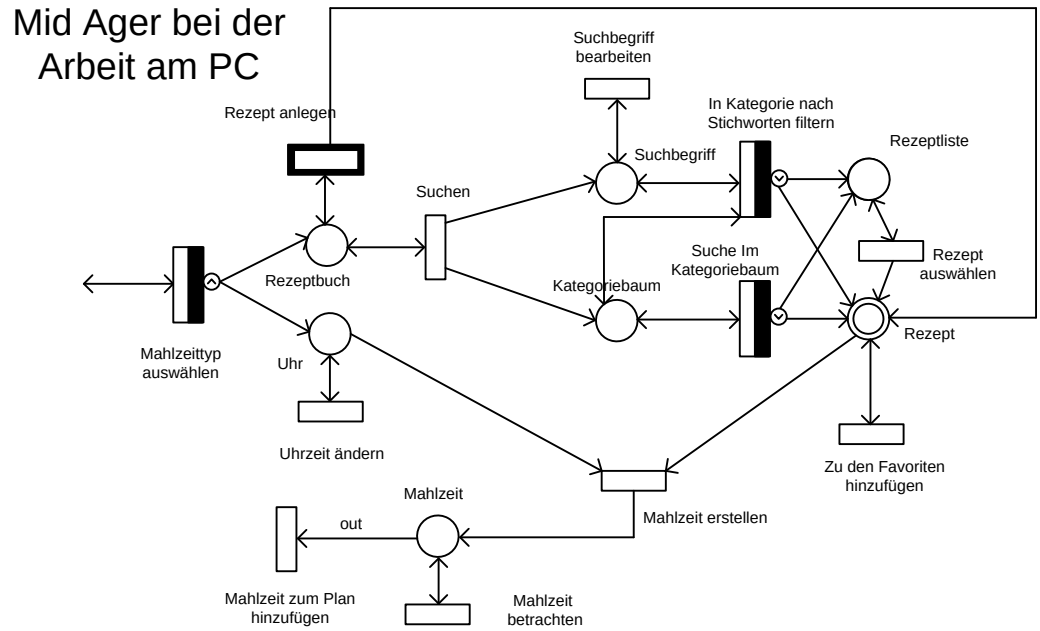
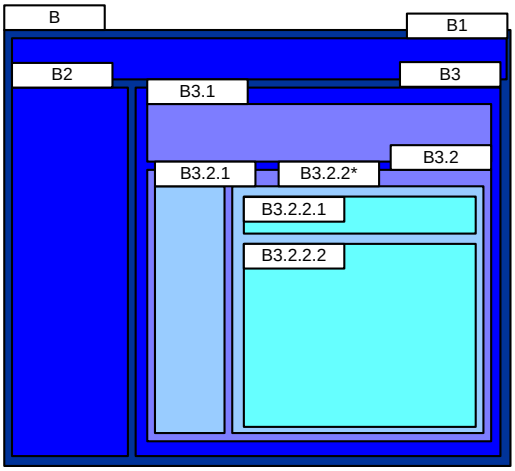


Abbildung 80: BSF-AIN „Mid Ager bei der Arbeit am PC“

9.2.4 Sequenzialisierung von parallelen Benutzerschnittstellen-fragmenten

Nach der Ermittlung des BSF-AINs muss überprüft werden, ob sich das generierte BSF-AIN auch auf das Layoutpattern des Nutzungsprofiles abbilden lässt. Hier ist eine Betrachtung des Layoutpattern und der ausgewählten BSF notwendig. Im Folgenden werden das Layoutpattern für das Nutzungsprofil „Mid Ager bei der Arbeit am PC“ und anschließend die im BSF-AIN identifizierten BSF mit ihrer Belegung der Bereiche des Layoutpattern präsentiert.



B = {B1, B2, B3}

B3 = {B3.1, B3.2}

B3.2 = {B3.2.1, B3.2.2}

B3.2.2* = {B3.2.2.1, B3.2.2.2}

Abbildung 81: Layoutpattern „Mid Ager bei der Arbeit am PC“

Abbildung 81 visualisiert das Layoutpattern für das Nutzungsprofil „Mid Ager bei der Arbeit am PC“. Die Beschreibung erfolgt analog zur Beschreibung des Layoutpattern für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ (vgl. Abschnitt 9.1.4).

Die beiden Layoutpattern unterscheiden sich in ihrer Struktur, so ist das in Abbildung 81 dargestellte Layoutpattern komplexer und spezifiziert insgesamt mehrere Bereiche. Begründet sind diese Unterschiede dadurch, dass Layoutpattern spezifisch für ein Endgerät definiert werden. Ein Bildschirm eines PC bietet eine viel größere Präsentationsfläche für die Benutzerschnittstelle als ein mobiles Endgerät. Folglich können auch mehr Bereiche angeboten und auch mehr BSF gleichzeitig dargestellt werden.

Eine weitere Besonderheit des Layoutpattern „Mid Ager bei der Arbeit am PC“ ist die Möglichkeit „dynamische Bereiche“ zu spezifizieren, die in Abbildung 81 durch das „*“-Symbol (B3.2.2*) gekennzeichnet sind. Solche dynamischen Bereiche werden beispielsweise durch neue Fenster, Pop-ups im Browser oder Alarmmeldungen realisiert. Sie ermöglichen es, beliebig viele Instanzen eines BSF gleichzeitig darzustellen. Dynamische Bereiche sind nicht modal. Das heißt, dass sie selbst und alle anderen Bereiche weiterhin erreichbar sind. Modale Dialoge, welche alle anderen blockieren, solange sie dargestellt werden, werden im Rahmen dieser Arbeit nicht betrachtet.

Nachdem das Layoutpattern definiert wurde, können jedem BSF des BSF-AIN Bereiche des Layoutpattern zugeordnet werden. Tabelle 21 visualisiert solch eine Belegung für das gegebene BSF-AIN (vgl. Abbildung 80).

Benutzerschnittstellenfragmente	B3			
	B.3.1	B3.2		
		B3.2.1	B3.2.2*	
			B3.2.2.1	B3.2.2.2
„Rezept anlegen“	■	■	■	■
„Suchen“	■			
„In Kategorie nach Stichworten suchen“		■		
„Suchbegriff bearbeiten“			■	
„Suche Im Kategoriebaum“		■		
„Rezept auswählen“			■	■
„Zu den Favoriten hinzufügen“			■	■
„Uhrzeit ändern“	■			
„Mahlzeit erstellen“			■	■
„Mahlzeit betrachten“	■	■	■	■
„Mahlzeit hinzufügen“	■	■	■	■

Tabelle 21: BSF des BSF-AINs „Mid Ager bei der Arbeit am PC“

Zusätzlich wurde eine Belegung der Bereiche für die Präsentationsfragmente, die durch die Stellen im BSF-AINs repräsentiert werden, in Tabelle 22 dargestellt.

Benutzerschnittstellenfragmente	B3			
	B.3.1	B3.2		
		B3.2.1	B3.2.2	
			B3.2.2.1	B3.2.2.2
„Rezeptbuch“	■			
„Rezeptliste“				■
„Rezept“			■	■
„Suchbegriff“			■	
„Kategoriebaum“		■		
„Uhr“	■			
„Mahlzeit“	■	■	■	■

Tabelle 22: PF des BSF-AIN „Mid Ager bei der Arbeit am PC“

Im Vergleich zum Nutzungsprofil „Youngster unterwegs mit dem PDA“ finden sich hier weniger BSF, die sich in den Bereichen überlappen, die sie belegen. Dennoch lassen sich Überlappungen identifizieren, die auch tatsächlich zu Konflikten führen können. Eine Analyse (vgl. Abschnitt 9.1.4) des BSF-AINs zeigt, dass die folgenden Konflikte auftreten können:

1. Die Stelle „Uhr“ und die Stelle „Rezeptbuch“ können nicht gleichzeitig dargestellt werden.
2. Wenn die Transition „Rezept anlegen“ durchgeführt wird, können die Stellen „Suchbegriff“, „Kategoriebaum“, „Rezeptliste“ und „Mahlzeit“ nicht markiert sein. Weiterhin dürfen auch die zu diesen Stellen zugehörigen Transitionen nicht aktiv sein.
3. Wenn die Transition „Suchen“ schaltet, darf die Stelle „Mahlzeit“ nicht markiert sein.
4. Nachdem die Transition „Mahlzeit erstellen“ geschaltet hat, dürfen die Stellen „Suchbegriff, „Kategoriebaum“ und „Rezeptliste“ nicht markiert sein. Weiterhin dürfen auch die zu diesen Stellen zugehörigen Transitionen nicht aktiv sein.

Da Konflikte identifiziert wurden, wird nun versucht, diese durch Sequenzialisierung aufzulösen. Hierzu werden, wie in Abschnitt 9.1.4 dargestellt, unterschiedliche Methoden zur Sequenzialisierung angewendet und überprüft, ob dadurch Konflikte behoben werden konnten.

Tabelle 23 präsentiert das Ergebnis des ersten Schrittes, in welchem die Methode 1 „Überschreiben von Interaktionsabläufen“ eingesetzt wird.

Transition	Start	Ende	Aufgelöste Konflikte
„Rezept anlegen“	{Suchbegriff}-	{Suchbegriff}+	Konflikt 2
	{Kategoriebaum}-	{Kategoriebaum}+	Konflikt 2
	{Mahlzeit}-	{Mahlzeit}+	Konflikt 2
	{Rezeptliste}-	{Rezeptliste}+	Konflikt 2
	[In Kategorien nach Stichworten suchen]-	[In Kategorien nach Stichworten suchen]+	Konflikt 2
	[Suchbegriff bearbeiten]-	[Suchbegriff bearbeiten]-	Konflikt 2
	[Rezept auswählen]-	[Rezept auswählen]+	Konflikt 2
	[Suche Im Kategoriebaum]-	[SucheImKategoriebaum]+	Konflikt 2
	[Mahlzeit erstellen]-	[Mahlzeit erstellen]+	Konflikt 2
	[Mahlzeit betrachten]-	[Mahlzeit betrachten]+	Konflikt 2
„Suchen“	{Mahlzeit}-		Konflikt 3
	[Mahlzeit betrachten]-		Konflikt 3
„Mahlzeit erstellen“	{Suchbegriff}-		Konflikt 4
	{Kategoriebaum}-		Konflikt 4
	{Rezeptliste}-		Konflikt 4
	[Suchen]-		Konflikt 4
	[In Kategorien nach Stichworten suchen]-		Konflikt 4
	[Suchbegriff bearbeiten]-		Konflikt 4
	[Rezept auswählen]-		Konflikt 4
	[SucheImKategoriebaum]-		Konflikt 4

Tabelle 23: Überschreiben von Interaktionsabläufen für „Mid Ager bei der Arbeit am PC“

Vergleicht man diese Ergebnisse mit dem Pendant für das Nutzungsprofil „Youngster unterwegs mit dem PDA“, so finden sich erstmalig für das BSF „Rezept anlegen“ Modifikationen, die nach der Ausführung des BSF durchgeführt werden (Spalte „Ende“ in Tabelle 23). Dies bedeutet, dass für die Dauer der Ausführung dieses BSF, die betreffenden Präsentationsfragmente und BSF überschrieben werden. Nach der Ausführung werden diese aber wiederhergestellt. Dies ist möglich, da die Stelle „Rezept“ in einem dynamischen Bereich dargestellt wird. Nach der Ausführung des BSF „Rezept anlegen“ wird dieses Präsentationsfragment aktiviert. Da es aber keinen Bereich belegt, der durch die vorher aktiven Präsentationsfragmente und BSF belegt war, können diese wiederhergestellt werden. Diese Form des Überschreibens war im Rahmen des Nutzungsprofiles „Youngster unterwegs mit dem PDA“ nicht möglich, da die Ausgangsstellen der entsprechenden Transitionen immer Bereiche belegt haben, welche auch von den vorhergehenden Präsentationsfragmenten und BSF benötigt wurden.

Nach der Anwendung dieser Methode verbleibt nur noch Konflikt 1. Dabei handelt es sich um die gleiche Problemstellung, wie sie an dieser Stelle auch für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ existierte. Die Präsentationsfragmente „Rezeptbuch“ und „Uhr“ können nicht gleichzeitig dargestellt werden. Beide müssen aber aktiviert sein, damit das BSF „Mahlzeit erstellen“ ausgeführt werden

kann. An dieser Stelle wird auf die Anwendung der einzelnen Methoden verzichtet, da dies zu dem gleichen Ergebnis (wie für das Nutzungsprofil „Youngster unterwegs mit dem PDA“) führt.

Die Methode „Auflösen durch Sequenzialisieren“ ist in der Lage diesen Konflikt zu beheben. Dabei können zwei Varianten (vgl. Abschnitt 9.1.4) identifiziert werden, die den Konflikt auflösen. Für das Nutzungsprofil „Mid Ager bei der Arbeit am PC“ wird nun die zweite Variante gewählt, bei der die Kante zwischen „Uhr“ und „Mahlzeit erstellen“ entfernt wird. Abbildung 82 präsentiert das BSF-AIN nach der Sequenzialisierung.

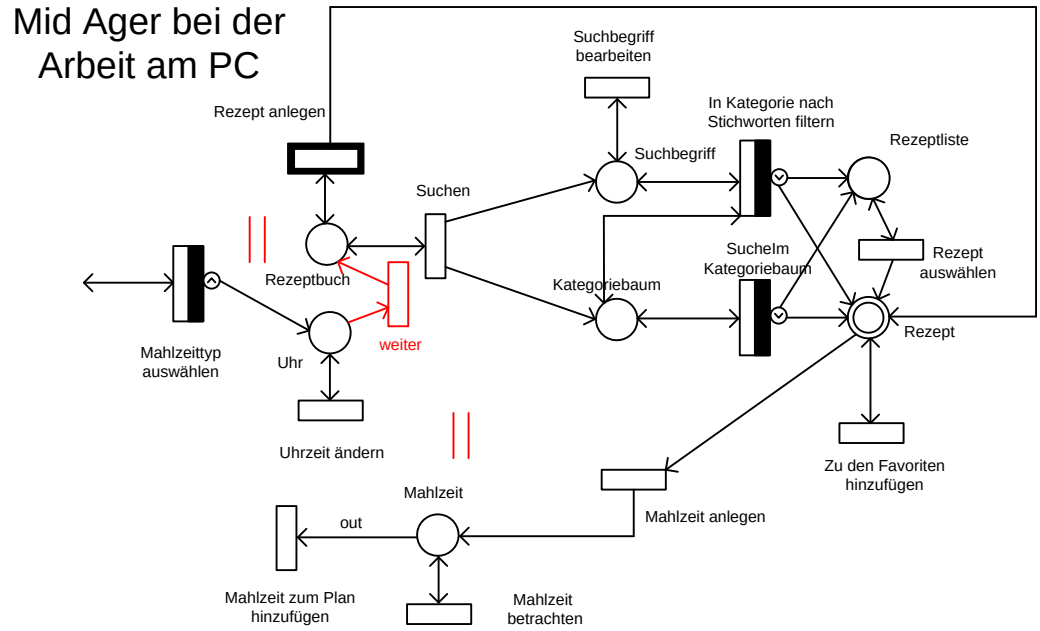


Abbildung 82: BSF-AIN „Mid Ager bei der Arbeit am PC“ nach der Sequenzialisierung

9.2.5 Komposition basierend auf dem BSF-AIN

Nachdem durch die Sequenzialisierung sichergestellt wurde, dass sich das BSF-AIN auf das Layoutpattern abbilden lässt, kann nun die Komposition der BSF durch die Intervallalgebra beschrieben werden. Aus dem in Abbildung 82 dargestellten BSF-AIN lassen sich die folgenden Relationen ableiten:

ALTERNATIVEN:

Uhr= ALT([Uhrzeit ändern], [weiter])

Rezeptbuch=ALT([Rezept anlegen], [Suchen])

Suchbegriff=ALT([Suchbegriff bearbeiten], [In Kategorien nach Stichworten filtern])

Kategoriebaum=ALT([SucheImKategoriebaum], [In Kategorien nach Stichworten filtern])

*Rezept=ALT([Zu den Favoriten hinzufügen], [Mahlzeit erstellen])**

Mahlzeit=ALT ([Mahlzeit betrachten], [Mahlzeit zum Plan hinzufügen])

STARTS:

Suchen= Suchbegriff STARTS Kategoriebaum

NETZ:

[Mahlzeittyp auswählen] OVERLAPS Uhr

[weiter] MEETS Rezeptbuch

[Rezept anlegen] OVERLAPS Rezept

[Suchen] OVERLAPS Suchen

[In Kategorien nach Stichworten Filtern] OVERLAPS ([Rezept auswählen] OR Rezept) IF ({Suchbegriff}.Status=aktiviert AND {Kategoriebaum}.Status=aktiviert)

[SucheImKategoriebaum] OVERLAPS ([Rezept auswählen] OR Rezept)

[Rezept auswählen] OVERLAPS Rezept

[Mahlzeit anlegen] Meets Mahlzeit

NEBENSTELLEN:

[Uhrzeit ändern] MEETS [Uhrzeit ändern]

[Suchbegriff bearbeiten] MEETS [Suchbegriff bearbeiten]

[Zu den Favoriten hinzufügen] MEETS [Zu den Favoriten hinzufügen]

[Mahlzeit betrachten] MEETS [Mahlzeit betrachten]

Die Ermittlung der Relation und die hier verwendete Syntax für die Beschreibung erfolgt analog zu Abschnitt 9.1.5. Im Vergleich zu dem dort präsentierten Pendant für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ finden sich in den hier dargestellten Relationen drei Besonderheiten: einerseits wird in der Relation *„[In Kategorien nach Stichworten Filtern] OVERLAPS ([Rezept auswählen] OR Rezept) IF ({Suchbegriff}.Status=aktiviert AND {Kategoriebaum}.Status=aktiviert)*“ erstmals eine Bedingung formuliert, die angibt ob das BSF ausgeführt werden kann. Solche Bedingungen müssen immer formuliert werden, wenn eine Transition über mehr als eine Eingangsstelle verfügt. Die Realisierung solcher Bedingungen erfolgt durch die Erweiterung der Startdialogfragmente des zugehörigen BSF.

Jedes der Startdialogfragmente ist in der Lage das BSF zu aktivieren, daher muss jedes um die Bedingung erweitert werden. Dies bedeutet, dass das BSF „In Kategorien nach Stichworten filtern“ sowohl durch die Markierung der Stelle „Kategoriebaum“ als auch der Stelle „Suchbegriff“ gestartet werden kann, wobei in beiden Fällen die Bedingung überprüft werden muss. Im gegebenen BSF-AIN wird die Bedingung allerdings immer erfüllt sein, da beide Stellen gleichzeitig durch die Transition „suchen“ markiert werden.

Die Starts-Relation, welche die zweite Besonderheit darstellt (im Nutzungsprofil „Youngster unterwegs mit dem PDA“ wurde diese nicht eingesetzt), beschreibt die gleichzeitige Aktivierung von zwei BSF. Realisiert wird diese, indem ein neues Startdialogfragment generiert wird, das die Ausführung der Startdialogfragmente der behandelten BSF realisiert.

Die dritte Besonderheit findet sich in der Alternativenrelation „*Rezept=ALT([Zu den Favoriten hinzufügen], [Mahlzeit erstellen])**“. Das „*“ gibt an, dass die beiden Alternativen in einem dynamischen Bereich ausgeführt werden. Das heißt, dass jedes Mal, wenn das Startdialogfragment, das durch die Alternative gebildet wurde, aufgerufen wird, ein neues Fenster generiert wird, in welchem die Alternativen ausgeführt werden. Um diese Art der Komposition durchführen zu können, müssen zwei Bedingungen erfüllt sein. Erstens muss die „Eingangsstelle“ des BSF im BSF-AIN durch eine doppelte Umrandung gekennzeichnet sein; zweitens muss der Bereich, in dem das BSF ausgeführt wird, auch im Layoutpattern als dynamischer Bereich gekennzeichnet sein. Doppelt umrandete Stellen im AIN geben an (vgl. Abbildung 82), dass diese Stellen mehr als eine Markierung enthalten dürfen. Ist es nun möglich dem Modell entsprechend auch mehr als eine Instanz dieses BSF gleichzeitig darzustellen (nur mit dynamischen Bereichen ist dies der Fall), so wird dies, wie in Anhang D.3 dargestellt, realisiert.

9.2.6 Diskussion der generierten Benutzerschnittstellen

Im Folgenden werden Ausschnitte aus dem Interaktionsablauf anhand der generierten Benutzerschnittstellen präsentiert. Im Fokus der Erläuterungen zu den folgenden Screenshots stehen Unterschiede im Vergleich zu der Benutzerschnittstelle für das Nutzungsprofil „Youngster unterwegs mit dem PDA“ und Eigenschaften, die dort noch nicht dargestellt werden konnten (bspw. dynamische Fenster).

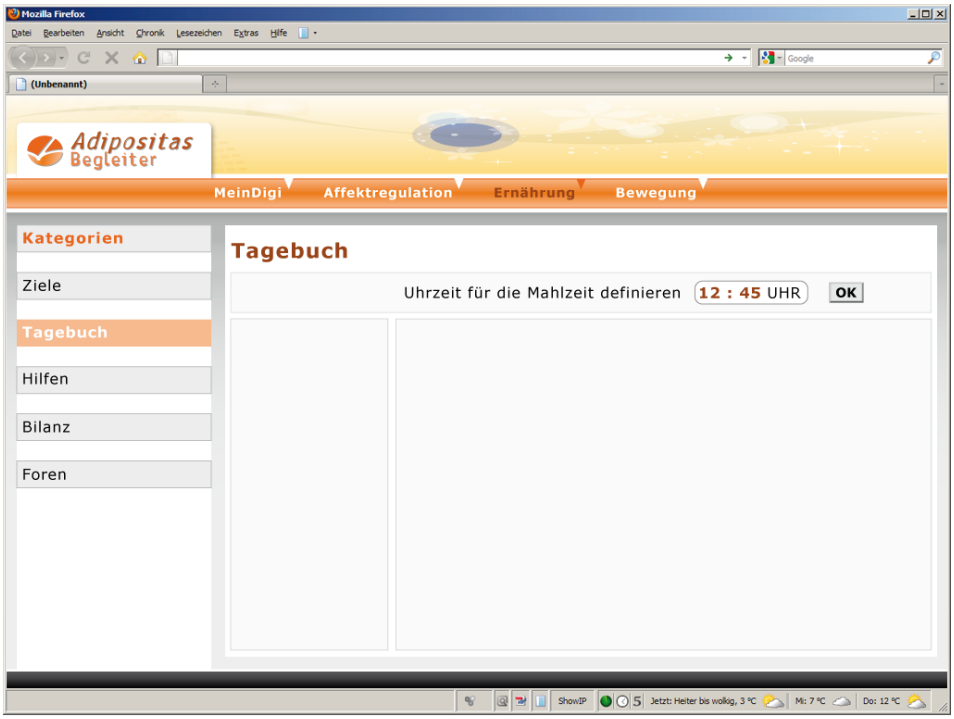


Abbildung 83: Screenshot: „Mid Ager bei der Arbeit am PC“-„Uhr“

Abbildung 83 visualisiert die Benutzerschnittstelle nach Ausführung des BSF „Mahlzeittyp auswählen“. Wie man der Abbildung entnehmen kann, ist nicht nur das Präsentationsfragment „Uhr“ dargestellt, sondern in den umgebenen Bereichen auch Menüs, wie sie als Dienstekategorien des Adipositas-Begleiters (vgl. Abbildung 11 in Abschnitt 3.3) vorgestellt wurden. Das obere, horizontale Menü in Abbildung 83 stellt die vier Kategorien des Adipositas-Begleiters dar, und die linke vertikale Liste Anwendungen, die einer dieser Kategorien (nämlich „Ernährung“) zugeordnet wurden.

Durch diese Abbildung wird veranschaulicht, dass die hier generierte Benutzerschnittstelle für die Interaktion „Mahlzeit anlegen“ selbst nur ein BSF der Anwendung „Adipositas-Begleiter“ ist. Als BSF belegt es eine Menge von Bereichen des Layoutpattern und kann durch Interaktionen, die nicht Teil des BSF-AINs sind, jederzeit überschrieben werden. Weiterhin zeigt dieses Beispiel, dass für die Ausführung komplexer BSF alle Bereiche, die benötigt werden, bereits beim Start bis zum Ende der Ausführung belegt werden. In Abbildung 83 ist beispielsweise der große Bereich rechts unten weitgehend leer, da dieser Bereich vom BSF „Mahlzeit erstellen“ belegt wird. Dieses Vorgehen ermöglicht nicht nur eine vereinfachte Betrachtung von BSF, sondern spiegelt auch wieder, dass dem Anwender die Durchführung von BSF als eine zusammengehörige Einheit präsentiert wird.

Nach der Generierung, der Evaluation und Nachbearbeitung der generierten Benutzerschnittstelle durch den Benutzerschnittstellen-Designer kann dieses BSF als Baustein in die „Domain Theory“ aufgenommen werden, und steht dort für zukünftige Anwendungen zur Verfügung.

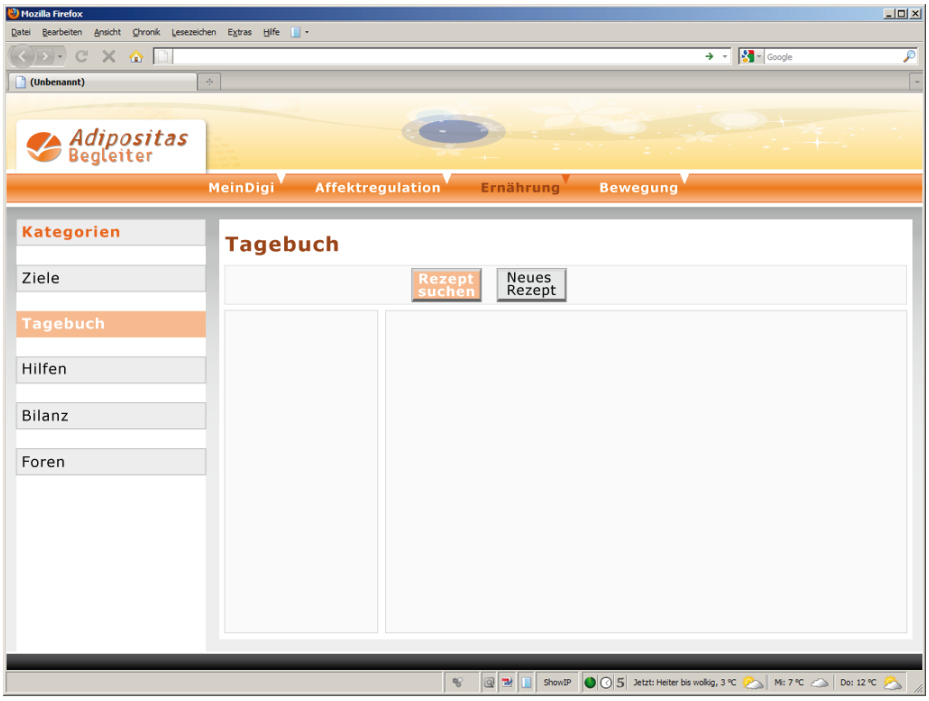


Abbildung 84: Screenshot: „Mid Ager bei der Arbeit am PC“-„Rezeptbuch“

Abbildung 84 zeigt die Benutzerschnittstelle, nachdem der Nutzer eine Uhrzeit festgelegt hat. Visualisiert ist das Präsentationsfragment „Rezeptbuch“, das als Menü realisiert wurde.

Abbildung 85: Screenshot: „Mid Ager bei der Arbeit am PC“-„Suchbegriff“ & "Kategoriebaum"

Abbildung 85 veranschaulicht die Benutzerschnittstelle, nachdem „Rezept suchen“ ausgewählt wurde. Aufgrund der Start-Relation werden gleichzeitig zwei Präsentationsfragmente („Kategoriebaum“ und „Suchbegriff“) sichtbar. Links wird der „Kategoriebaum“ präsentiert, während im horizontalen Bereich das Präsentationsfragment „Suchbegriff“ dargestellt wird. Die Darstellung von BSF, die über eine Start-Relation verknüpft wurden, erfordert keine besonderen Maßnahmen für die Präsentation. Beide Präsentationsfragmente belegen unterschiedliche Bereiche des Layoutpattern und auch die dazugehörigen BSF überschneiden sich nicht, was durch den Arbeitsschritt der Sequenzialisierung sichergestellt wurde.

Abbildung 86: Screenshot: „Mid Ager bei der Arbeit am PC“-„Rezeptliste“

Abbildung 86 zeigt die Rezeptliste (mittlerer Bereich) als das Ergebnis der Suche. Gesucht wird, indem der Benutzer eine Kategorie auswählt und alle Rezepte dieser Kategorie in der Rezeptliste angezeigt werden. Anschließend hat er die Möglichkeit, mit Hilfe eines Suchbegriffes zu filtern. Wechselt der Benutzer die Kategorie, so wird die Rezeptliste mit den Rezepten der neu gewählten Kategorie überschrieben. Damit der Filter wieder aktiviert wird, muss der Benutzer den Suchbegriff bestätigen.

Abbildung 87: Screenshot: „Mid Ager bei der Arbeit am PC“-„Rezept“

Abbildung 87 zeigt einen Screenshot der Benutzerschnittstelle, nachdem ein Rezept der Rezeptliste ausgewählt wurde. Dargestellt ist das Präsentationsfragment „Rezept“, das innerhalb eines Pop-up Fensters angezeigt wird. Das Pop-up Fenster kann vom Anwender verschoben oder geschlossen werden. Auch wenn diese Interaktionsmöglichkeiten nicht explizit modelliert wurden, so bietet jedes Pop-up Fenster diese Möglichkeiten.

Der Anwender hat die Möglichkeit, weitere Pop-up Fenster mit unterschiedlichen Rezepten zu öffnen und parallel dazu weiter im Rezeptbuch zu suchen.

Abbildung 88: Screenshot: „Mid Ager bei der Arbeit am PC“-„Mahlzeit“

Abbildung 88 zeigt einen Screenshot, nachdem in einem der geöffneten Pop-up Fenster „Mahlzeit erstellen“ gewählt wurde. Dargestellt ist das Präsentationsfragment „Mahlzeit“. Nur das Rezept, für das „Mahlzeit erstellen“ aufgerufen wurde, wird dabei geschlossen. Sollten weitere Rezepte als Pop-up Fenster verfügbar sein, so bleiben diese bestehen. Wird in einem dieser Fenster „Mahlzeit erstellen“ aufgerufen, so wird das aktuelle Präsentationsfragment, das die Mahlzeit darstellt, überschrieben.

Durch das BSF-„Mahlzeit zum Plan hinzufügen“ wird die Ausgangsstelle des Netzes erreicht. Das Netz ist damit abgeschlossen und alle Präsentationsfragmente und noch aktiven BSF werden beendet.

9.3 „Best Ager zuhause am Fernseher“

Im letzten Schritt der Evaluation wird der Generierungsprozess einer Benutzerschnittstelle für den Nutzungskontext „Best Ager zuhause am Fernseher“ beschrieben.

Im Vergleich zu den beiden vorhergehenden Nutzungsprofilen stellt insbesondere die Wahl des Endgerätes, nämlich ein Fernseher, der über eine Fernbedienung gesteuert wird, ganz spezielle Anforderungen an die zu generierende Benutzerschnittstelle. Zielsetzung ist es darzustellen, wie das entwickelte Verfahren mit den stringenten Restriktionen, die sich dadurch ergeben, umgeht und welche Ergebnisse produziert werden.

Für die Präsentation des Generierungsprozesses wurde bewusst dieselbe Kapitelstruktur wie bei den beiden vorherigen Generierungsdurchläufen gewählt, um Unterschiede in den einzelnen Schritten des Generierungsprozesses aufzudecken, und darzustellen, wie sich die einzelnen Methoden angewendet auf unterschiedliche Nutzungsprofile verhalten.

9.3.1 Modellierung der Interaktionsbeschreibung

Die Interaktionsbeschreibung spezifiziert die Interaktion des Benutzers, ohne Aussagen über das verwendete Endgerät, den Nutzer und die Umgebung der Nutzung zu treffen. Sie spezifiziert somit auf einer abstrakten Ebene die zu generierende Benutzerschnittstelle und ist für alle Nutzungskontexte gleich.

9.3.2 Modellierung des Nutzungskontextes und Generierung des Nutzungsprofils

Wie auch für die beiden vorhergehenden Nutzungsprofile ist für die Generierung der Benutzerschnittstelle eine Spezifikation eines Nutzungskontextes, wie er in Tabelle 24 präsentiert wird, ausreichend. In Abschnitt 6.4 wurde ein Nutzungskontextmodell für die Anwendungsdomäne des „Adipositas-Begleiters“ eingeführt. Dieses bildet die Grundlage für die Spezifikation der Nutzungskontexte und schließlich für die Generierung der Nutzungsprofile.

	Umgebung	Nutzer	Endgerät
Default-Profil	Zuhause	Nicht technikaffiner Nutzer	SetTop-Box
Kontext-Element	- Lichtverhältnisse: ++ - Personen im Umfeld: + - Ablenkung: -	- Sehvermögen: o - Motorik: -	- Tastatur

Tabelle 24: Nutzungskontext „Best Ager zuhause am Fernseher“

Auf die Beschreibung der einzelnen Kontextelemente des Nutzungskontextes „Best Ager zuhause am Fernseher“ wird verzichtet. Diese findet sich in Abschnitt 6.4 dieser Arbeit. Stattdessen werden die Unterschiede und Besonderheiten zu den beiden vorhergegangenen Nutzungskontexten anhand der drei Kontextkomponenten diskutiert:

- Kontextkomponente Umgebung
  Das Defaultprofil „Zuhause“ beschreibt die Nutzung einer Anwendung in einem häuslichen Umfeld. Im Gegensatz zum Defaultprofil „Unterwegs“ und „Arbeit“ beschreibt es eine private Umgebung, so dass Personen im Umfeld oder auch andere Ablenkungen nicht berücksichtigt werden müssen.
  Für die Benutzerschnittstelle bedeutet dies, dass auch komplexe Interaktionsobjekte angeboten werden können. Weiterhin kann sich die Benutzerschnittstelle auch dynamischer Interaktionsabläufe (beispielsweise parallele Darstellung von Teilen der Benutzerschnittstelle, die über unterschiedliche Fenster verteilt sind) bedienen. Multimediale Inhalte (auch Audio Inhalte) werden präferiert, da unterbrechungsfreie Bedingungen gegeben sind.

- Kontextkomponente Nutzer
  Das Defaultprofil „Nicht technikaffiner Nutzer“ beschreibt eine Person, die nicht mit dem Endgerät und auch nicht mit den für dieses Endgerät typischen Benutzerschnittstellen vertraut ist. Es sollte immer versucht werden, möglichst einfach zu bedienende Interaktionsobjekte zu wählen und komplexe Interaktionsobjekte gänzlich zu meiden. Grundsätzlich wird ein sequentieller Interaktionsablauf gewünscht, der den Benutzer schrittweise durch den Interaktionsprozess führt. Dies gilt sogar dann, wenn der Interaktionsablauf dadurch erheblich verlängert wird. Eine effiziente Durchführung oder die Möglichkeit zwischen Alternativen (zur Anwendungsdurchführung) wählen zu können, wird nicht priorisiert.
  Für die Benutzerschnittstelle bedeutet dies, dass eine niedrige Informationsdichte von Benutzerschnittstellen erwünscht ist. Weiterhin sollte Parallelität von Interaktionsabläufen vermieden werden. Multimediale Inhalte wie Video und Audio sind hingegen erwünscht.
- Kontextkomponente Endgerät
  Das Endgerät Fernseher definiert besondere Rahmenbedingungen als Benutzerschnittstelle. Zwar verfügt es über einen großen und (bei aktuellen Geräten) auch hochauflösenden Bildschirm. Es ist allerdings zu berücksichtigen, dass durch die Entfernung des Anwenders zum Endgerät die Präsentation entsprechend groß sein muss, um überhaupt lesbar zu sein. Tatsächlich können nicht mehr Inhalte als auf einem PDA für den Nutzer gut sichtbar dargestellt werden. Bei der Darstellung von Text gestaltet sich dies noch schwieriger. Durch die Entfernung ist das konzentrierte Lesen langer Texte nur schwer möglich, wie man am Beispiel des klassischen Videotextes sehen kann. Eine weitere Einschränkung definiert die Fernbedienung als Interaktionsmedium. Tatsächlich erfolgt die Navigation primär über die vier Steuertasten (rauf, runter, link und rechts) und eine Bestätigungstaste. Insbesondere Freitext- und Ziffern-Eingaben sind daher nur schwer zu realisieren.

Der Nutzungskontext „Best Ager zuhause am Fernseher“ beschreibt Anforderungen an eine Benutzerschnittstelle, die sehr strikten Restriktionen unterliegen. Durch die Wahl des Endgerätes, das nur rudimentär über eine Fernbedienung gesteuert werden kann, verbunden mit der Notwendigkeit die Präsentation so groß zu gestalten, dass sie noch lesbar bleibt, wird die Menge an möglichen Interaktionsobjekten stark eingeschränkt. Erschwerend kommt hinzu, dass der Anwender selbst nur rudimentäre Kenntnisse im Umgang mit dem Gerät mitbringt, und weiterhin durch die Interaktionen geführt werden will.

9.3.3 Generierung des BSF-AINs

Die Generierung des BSF-AINs nutzt die Interaktionsbeschreibung als Ausgangslage, und ermittelt eine passende Abbildung auf die BSF der „Domain Theory“ mit Hilfe des OM-Kataloges und der Bewertungsfunktion, die sich des Nutzungsprofils und der „Domain Theory“ bedient.

Um die Vergleichbarkeit der Generierungsdurchläufe mit den beiden vorhergehenden Nutzungsprofilen zu gewährleisten, erfolgt die Diskussion der Generierung des BSF-AINs anhand eines Baumes, wie er in Abschnitt 9.1.3 für „Youngster unterwegs mit dem PDA“ und in Abschnitt 9.2.3 für „Mid Ager bei der Arbeit am PC“ beschrieben wurde.

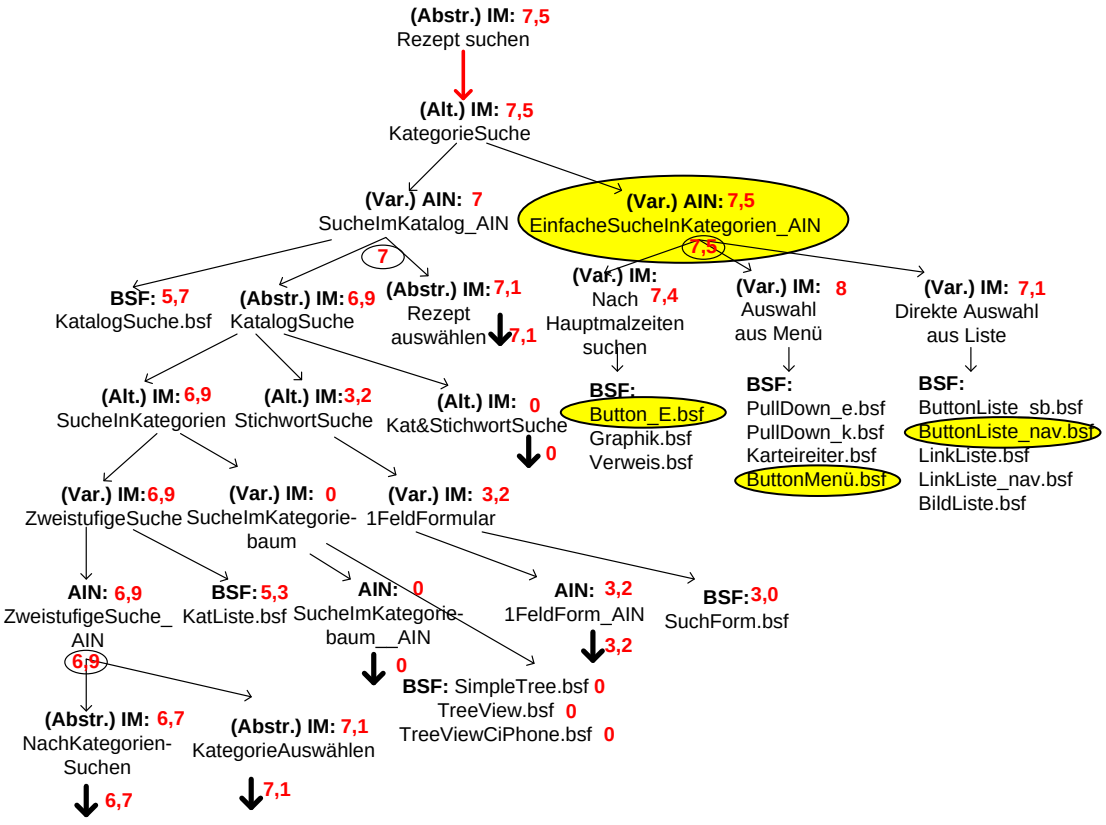


Abbildung 89: Ermittlung des BSF-AINs – „Best Ager zuhause am Fernseher“

Der in Abbildung 89 dargestellte Baum, gleicht in der Struktur dem Baum der für das Nutzungsprofil „Mid Ager bei der Arbeit am PC“ erstellt wurde. Auch für den Nutzungskontext „Best Ager zuhause am Fernseher“ wurde für die „Abstrakte Interaktion“ „KatalogSuche“ keine Alternativen komponiert, da aus dem Nutzungsprofil hervorgeht, dass Variationsmöglichkeiten im Interaktionsfluss nicht erwünscht sind. Auf eine genauere Betrachtung der Alternative „Kat&StichwortSuche“ wurde ebenfalls verzichtet, da diese Alternative auf dem gegebenen Nutzungsprofil nicht realisierbar ist. Stattdessen wurde die Variante „EinfacheSucheInKategorien_AIN“ genauer aufgeschlüsselt, die durch die Bewertungsfunktion als beste Realisierung identifiziert wurde.

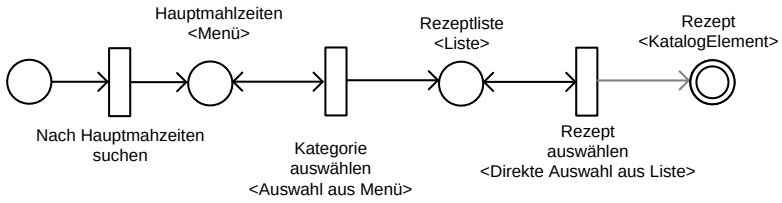


Abbildung 90: Ausschnitt aus dem OM-Katalog: EinfacheSucheInKategorien_AIN

Abbildung 90 präsentiert einen Ausschnitt aus dem OM-Katalog (vgl. Tabelle 14), der das AIN „EinfacheSucheInKategorien_AIN“ visualisiert. Der Ausschnitt wird an dieser Stelle genauer betrachtet, da gezeigt werden soll, dass in AINs nicht nur „Abstrakte Interaktionen“ für die Modellierung verwendet werden. Das AIN „EinfacheSucheInKategorien“ verwendet Varianten und Alternativen, da der Modellierer bereits konkretere Vorstellungen über die mögliche Realisierung geben möchte. Bei den in

Abbildung 90 benutzten Interaktionsmetaphern handelt es sich jeweils um Varianten, wie auch aus dem Baum in Abbildung 89 ersichtlich ist. Die Verwendung von Varianten ermöglicht bereits eine nahe an der tatsächlichen Realisierung liegende Beschreibung des AINs. So wird in „SucheInKategorien_AIN“ bereits vorgegeben, wie BSF (bspw. Hauptmahlzeiten als Menü, Rezeptlisten als Listen, usw.) in einer Benutzerschnittstelle realisiert werden. Dabei definiert eine Variante immer noch nicht die konkrete Realisierung. Aus dem Baum in Abbildung 89 lässt sich entnehmen, dass Varianten durch unterschiedliche BSF in der „Domain Theory“ realisiert werden können, so dass der Bewertungsalgorithmus auch hier (wenn auch eingeschränkt) zum tragen kommt.

Abbildung 91 präsentiert das generierte BSF-AIN, wobei auf die Angabe der ausgewählten BSF verzichtet wurde, um das BSF-AIN für den Leser verständlicher zu gestalten.

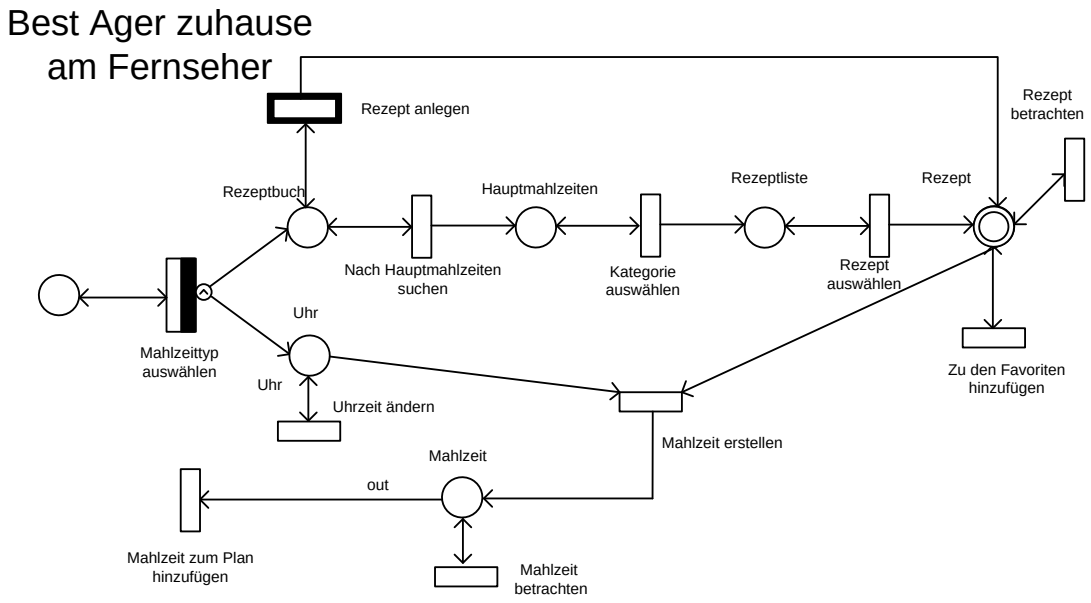
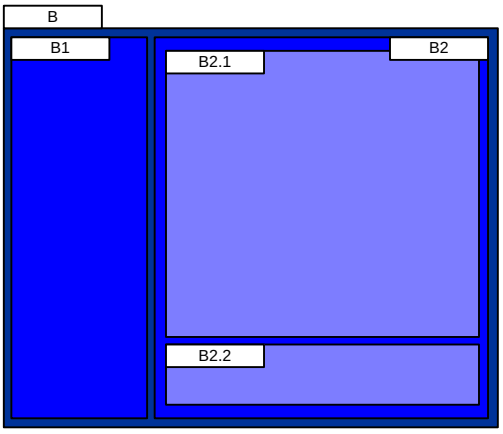


Abbildung 91: BSF-AIN – „Best Ager zuhause am Fernseher“

9.3.4 Sequenzialisierung von parallelen Benutzerschnittstellen-fragmenten

Im nächsten Generierungsschritt wird überprüft, ob sich das generierte BSF-AIN auch auf das Layoutpattern des Nutzungsprofils abbilden lässt. Hier ist eine Betrachtung des Layoutpattern und der ausgewählten BSF notwendig. Im Folgenden werden das Layoutpattern für das Nutzungsprofil „Best Ager zuhause am Fernseher“ und anschließend die im BSF-AIN identifizierten BSF mit ihrer Belegung der Bereiche des Layoutpatterns präsentiert.



B = {B1, B2}

B2 = {B2.1, B2.2}

Abbildung 92: Layoutpattern Set-Top-Box

Das Nutzungsprofil „Best Ager zuhause am Fernseher“ ist sehr einfach strukturiert, da die zur Verfügung stehende Präsentationsfläche (trotz großer Displaygröße) sehr beschränkt ist.

Benutzerschnittstellenfragmente	B1	B2	
		B2.1	B2.2
„Rezept anlegen“			
„Nach Hauptmahlzeiten suchen“			
„Kategorie auswählen“			
„Rezept auswählen“			
„Rezept betrachten“			
„Uhrzeit ändern“			
„Mahlzeit erstellen“			
„Mahlzeit betrachten“			
„Mahlzeit hinzufügen“			
„Zu den Favoriten hinzufügen“			

Tabelle 25: BSF des BSF-AINs „Best Ager zuhause am Fernseher“

Zusätzlich wurde noch eine Belegung der Bereiche für die Präsentationsfragmente, die durch die Stellen im BSF-AINs repräsentiert werden, in Tabelle 26 dargestellt.

Präsentationsfragmente	B1	B2	
		B2.1	B2.2
„Rezeptbuch“			
„Hauptmahlzeiten“			
„Rezeptliste“			
„Rezept“			
„Uhr“			
„Mahlzeit“			

Tabelle 26: PF des BSF-AIN „Best Ager zuhause am Fernseher“

Aus den beiden vorhergegangenen Tabellen kann leicht ermittelt werden, dass eine Vielzahl von Konflikten in der Ausführung des BSF-AIN auftreten werden. Viele Bereiche überschneiden sich, so dass eine starke Sequenzialisierung der Abläufe erforderlich wird.

Im Einzelnen können die folgenden Konflikte identifiziert werden:

1. Die Stellen „Rezeptbuch“ und „Uhr“ können nach Ausführung der Transition „Mahlzeittyp auswählen“ nicht gleichzeitig markiert werden.
2. Wenn „Rezept anlegen“ ausgeführt wird, dürfen außer „Rezeptbuch“ und „Rezept“ keine Stellen markiert und keine Transitionen aktiviert sein.

3. Wenn „Nach Hauptmahlzeiten suchen“ ausgeführt wird, dürfen außer „Rezeptbuch“ und „Hauptmahlzeit“ keine Stellen markiert und keine Transitionen aktiviert sein.
4. Nachdem „Rezept anlegen“ geschaltet hat, dürfen die Stelle „Rezeptbuch“ und „Rezept“ nicht gleichzeitig markiert sein.
5. Nachdem „Kategorie auswählen“ geschaltet hat, dürfen die Stellen „Rezeptbuch“, „Rezept“ und „Mahlzeit“ nicht mehr markiert sein. Weiterhin dürfen die Transitionen „Rezept anlegen“, „Nach Hauptmahlzeiten suchen“, „Rezept betrachten“, „Zu den Favoriten hinzufügen“, “Mahlzeit erstellen“ und „Mahlzeit betrachten“ nicht aktiviert sein.
6. Wenn „Rezept betrachten“ oder „Zu den Favoriten hinzufügen“ schaltet, dürfen die Stellen „Uhr“, „Rezeptbuch“, „Rezeptliste“ und „Mahlzeit“ nicht markiert sein. Weiterhin dürfen die Transitionen „Uhrzeit ändern“, „Rezept anlegen“, „Nach Hauptmahlzeiten suchen“, „Kategorie auswählen“, „Rezept auswählen“ und „Mahlzeit betrachten“ nicht aktiviert sein.
7. Wenn „Mahlzeit erstellen“ schaltet, dürfen die Stellen „Rezeptbuch“, „Rezeptliste“ und „Mahlzeit“ nicht markiert sein. Weiterhin dürfen die Transitionen „Rezept anlegen“, „Nach Hauptmahlzeiten suchen“, „Kategorie auswählen“, „Rezept auswählen“ und „Mahlzeit betrachten“ nicht aktiviert sein.
8. Wenn „Mahlzeit betrachten“ schaltet, dürfen die Stellen „Rezeptbuch“, „Rezeptliste“ und „Rezept“ nicht markiert sein. Weiterhin dürfen die Transitionen „Rezept anlegen“, „Nach Hauptmahlzeiten suchen“, „Kategorie auswählen“ und „Rezept auswählen“, „Rezept betrachten“ und zu den „Favoriten hinzufügen“ nicht aktiviert sein.
9. Nachdem „Rezept auswählen“ geschaltet hat, dürfen die Stellen „Rezept“ und „Rezeptliste“ nicht gleichzeitig markiert sein.

Tabelle 27 präsentiert das Ergebnis des ersten Schrittes, in welchem die Methode 1 „Überschreiben von Interaktionsabläufen“ eingesetzt wird.

Transition	**Start**	**Ende**	**Aufgelöste Konflikte**
„Nach Hauptmahlzeiten suchen“	{Rezeptliste}-		Konflikt 3
	{Rezept}-		Konflikt 3
	{Uhr}-		Konflikt 3
	{Mahlzeit}		Konflikt 3
	[Uhrzeit ändern]-		Konflikt 3
	[Kategorie auswählen]-		Konflikt 3
	[Rezept auswählen]-		Konflikt 3
	[Rezept betrachten]-		Konflikt 3
	[Zu den Favoriten hinzufügen]-		Konflikt 3
	[Mahlzeit erstellen]-		Konflikt 3
	[Mahlzeit betrachten-]		Konflikt 3
„Kategorie auswählen“	{Rezeptbuch}-		Konflikt 5
	{Rezept}-		Konflikt 5
	{Uhr}-		Konflikt 5

	{Mahlzeit}- [Uhrzeit ändern]- [Rezept anlegen]- [Nach Hauptmahlzeiten suchen]- [Rezept auswählen]- [Rezept betrachten]- [Zu den Favoriten hinzufügen]- [Mahlzeit erstellen]- [Mahlzeit betrachten-]		Konflikt 5 Konflikt 5 Konflikt 5 Konflikt 5 Konflikt 5 Konflikt 5 Konflikt 5 Konflikt 5 Konflikt 5
„Rezept betrachten“ & „Zu den Favoriten hinzufügen“	{Rezeptliste}- {Uhr}- {Mahlzeit}- [Uhrzeit ändern]- [Kategorie auswählen]- [Rezept auswählen]- [Mahlzeit betrachten-]		Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6 Konflikt 6
„Mahlzeit erstellen“	{Rezeptliste}- {Mahlzeit}- [Kategorie auswählen]- [Rezept auswählen]- [Mahlzeit betrachten-]		Konflikt 7 und 8 Konflikt 7 und 8 Konflikt 7 und 8 Konflikt 7 und 8 Konflikt 7 und 8

Tabelle 27: Überschreiben von Interaktionsabläufen für „Best Ager zuhause am Fernseher“

Die Vielzahl der in Tabelle 27 präsentierten Erweiterungen begründet sich dabei nur teilweise auf der Menge der aufzulösenden Konflikte. Vielmehr führen Konflikte, die durch das Überschreiben gar nicht aufgelöst werden können (im Beispiel „Rezept anlegen“ und „Rezept wählen“), zu der dargestellten Fülle.

Dabei handelt es sich offensichtlich um eine Schwäche des Algorithmus, der die Anzahl der Knoten im Interaktionsbaum, die nicht auf das Layoutpattern abgebildet werden können, als Erfolgskriterium wählt. Das Überschreiben ermöglicht es, die Anzahl solcher Knoten zu reduzieren, ohne aber die Ursache für den Konflikt zu lösen.

"Rezept auswählen" beispielsweise markiert die Stelle „Rezept“ und „Rezeptliste“ (Konflikt 7). Im Interaktionsbaum werden aber alle Interaktionspfade abgebildet, unabhängig, ob ein Konflikt identifiziert wurde oder nicht. Folglich kann beispielsweise durch die Erweiterung der Transition „Zu den Favoriten hinzufügen“ die Markierung aus der Stelle „Rezeptliste“ entfernt werden, so dass der daraus resultierende Knoten wieder auf das Layoutpattern abbildbar ist.

Durch eine Anpassung des Algorithmus können solche „überflüssigen“ Erweiterungen verhindert werden. Eine solche Anpassung wird sich dabei nicht auf die zu generierende Benutzerschnittstellen auswirken, da durch die notwendige Sequenzialisierung der Nebenstellen (Abbildung 93) die Erweiterungen überflüssig werden, und die zu überschreibenden Stellen und Transitionen nicht mehr markiert sein können.

Auf eine Durchführung der übrigen Schritte der Sequenzialisierung wird verzichtet, da diese im Vergleich zu den beiden vorhergegangenen Durchläufen keine neuen Erkenntnisse bringen.

In Abbildung 93 wird das Ergebnis der Sequenzialisierung präsentiert.

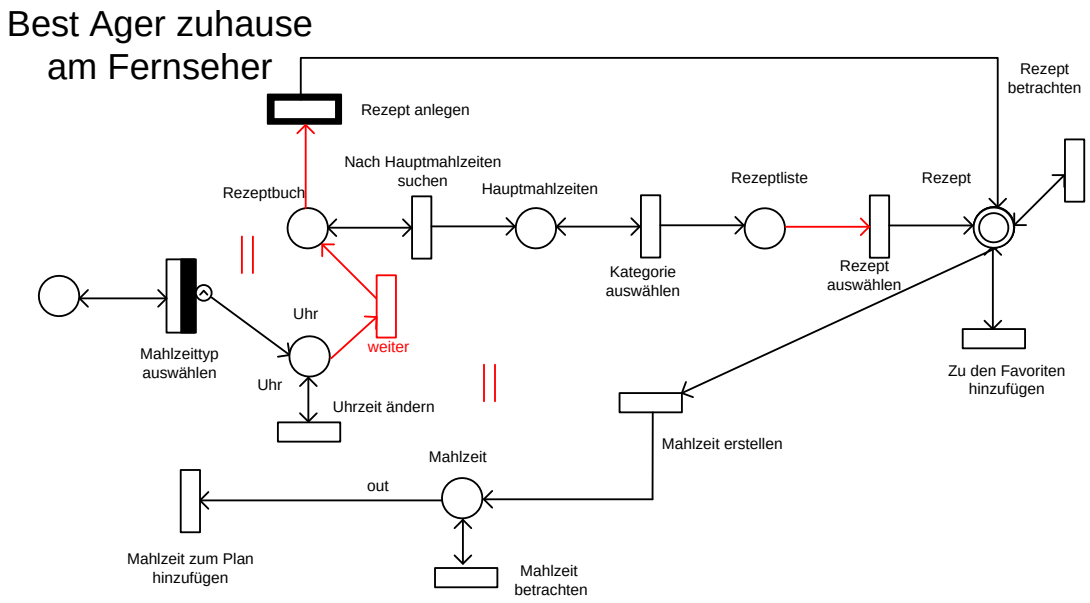


Abbildung 93: BSF-AIN - „Best Ager zuhause am Fernseher“ nach der Sequenzialisierung

9.3.5 Komposition basierend auf dem BSF-AIN

Folgende Relationen lassen sich aus dem in Abbildung 93 dargestellten BSF-AIN ableiten:

ALTERNATIVEN:

Uhr= ALT([Uhrzeit ändern], [weiter])

Rezeptbuch=ALT([Rezept anlegen], [Nach Hauptmahlzeiten suchen])

Rezept=ALT([Zu den Favoriten hinzufügen], [Mahlzeit erstellen], [Rezept betrachten])

Mahlzeit=ALT ([Mahlzeit betrachten], [Mahlzeit zum Plan hinzufügen])

NETZ:

[Mahlzeittyp auswählen] OVERLAPS Uhr

[weiter] MEETS Rezeptbuch

[Rezept anlegen] OVERLAPS Rezept

[Nach Hauptmahlzeiten suchen] OVERLAPS ([Kategorie wählen])

[Kategorie auswählen] MEETS [Rezept auswählen]

[Rezept auswählen] MEETS Rezept

[Mahlzeit erstellen] MEETS Mahlzeit

NEBENSTELLEN:

[Uhrzeit ändern] MEETS [Uhrzeit ändern]

[Rezept betrachten] MEETS [Rezept betrachten]

[Zu den Favoriten hinzufügen] MEETS [Zu den Favoriten hinzufügen]

[Mahlzeit betrachten MEETS [Mahlzeit betrachten]]

9.3.6 Diskussion der generierten Benutzerschnittstellen

Im Folgenden werden Ausschnitte aus dem Interaktionsablauf anhand der generierten Benutzerschnittstellen präsentiert.

Abbildung 94: Screenshot: „Best Ager zuhause am Fernseher“-„Uhr“

Abbildung 94 präsentiert den Screenshot der generierten Benutzerschnittstelle nach Ausführung des BSF „Mahlzeittyp auswählen“. Dargestellt wird das Präsentationsfragment „Uhr“ im rechten Bereich des Screenshots.

Bedient wird die Benutzerschnittstelle über eine Fernbedienung, indem der Benutzer über die vier Steuertasten (rauf, runter, link und rechts) auf einen der 6 Buttons navigiert und mit der „Bestätigungstaste“ den Button auslöst.

Als technische Realisierung wird eine Set-Top-Box verwendet, die in der Lage ist, einen stark vereinfachten Web-Browser darzustellen, welcher sowohl die Präsentation als auch die Steuerung mittels einer Fernbedienung umsetzt.

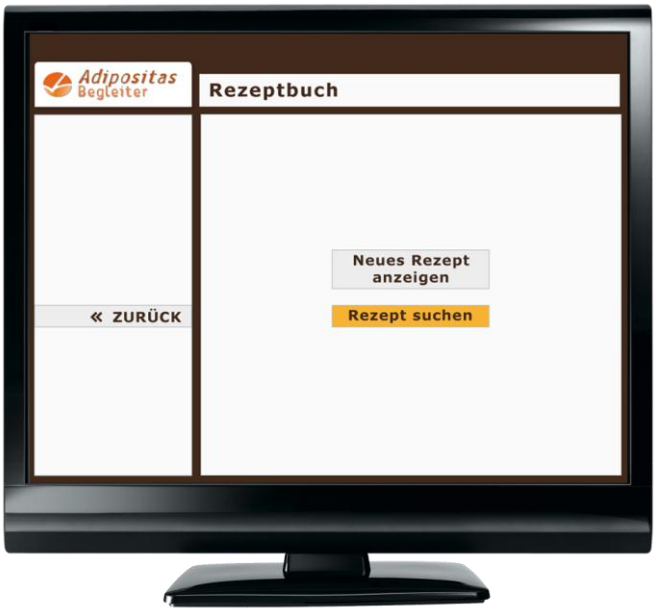


Abbildung 95: Screenshot: „Best Ager zuhause am Fernseher“-„Rezeptbuch“

Abbildung 95 zeigt einen Screenshot des Präsentationsfragments „Rezeptbuch“, nachdem eine Uhrzeit festgelegt wurde.

Abbildung 96: Screenshot: „Best Ager zuhause am Fernseher“-„Hauptmahlzeiten“

Abbildung 96 zeigt die Benutzerschnittstelle, nachdem „Rezept suchen“ ausgeführt wurde. Die Stelle Hauptmahlzeiten wird im linken Bereich als vertikale Button-Liste dargestellt.

Der Benutzer navigiert zwischen dem linken Bereich („Hauptmahlzeiten“) und dem rechten Bereich („Rezeptbuch“) über die Steuertasten „links“ und „rechts“.

Abbildung 97: Screenshot: „Best Ager zuhause“-„Rezeptliste“

Abbildung 97 zeigt die Benutzerschnittstelle, nachdem eine der Hautmahlzeiten ausgewählt wurde. Dargestellt wird eine Kategorieliste.

Abbildung 98: Screenshot: „Best Ager zuhause am Fernseher“-„Rezeptliste“

Abbildung 98 zeigt eine Seite der Rezeptliste. Wenn mehr als 9 Rezepte in einer Kategorie enthalten sind, werden die Buttons „vor“ und „zurück“ aktiviert. Mit Hilfe dieser Buttons kann der Benutzer die einzelnen Seiten der Rezeptliste erreichen.

Abbildung 99: Screenshot: „Best Ager zuhause am Fernseher“-„Rezept“

Abbildung 99 zeigt einen Screenshot, nachdem ein Rezept der Rezeptliste ausgewählt wurde. Dargestellt ist das Präsentationsfragment „Rezept“.

Abbildung 100: Screenshot „Best Ager zuhause am Fernseher“-„Mahlzeit“

Abbildung 100 zeigt einen Screenshot, nachdem „Mahlzeit erstellen“ ausgewählt wurde. Durch den Button „Mahlzeit erstellen“ wird die Mahlzeit dem Rezeptbuch hinzugefügt und das BSF-AIN wird beendet.

9.4 Status der Evaluation und weitere Schritte zum Nachweis des Verfahrens

Zielsetzung dieses Kapitels war die praktische Anwendung des entwickelten Verfahrens zur Generierung von Benutzerschnittstellen als ein „Proof of Concept“ der Lösung.

Ausgangslage für diese Untersuchung bildete eine Interaktionsbeschreibung aus der Domäne des „Adipositas-Begleiters“, die die Benutzerschnittstelle für die Aktivität „Mahlzeit erstellen“ spezifiziert. Weiterhin wurden drei Nutzungskontexte definiert, die möglichst unterschiedliche Anforderungen an die zu generierenden Benutzerschnittstellen stellen sollten. Für jeden der Nutzungskontexte wurde das in der Arbeit entwickelte Verfahren angewendet und eine ausführbare Benutzerschnittstelle generiert. Dabei wurde das Verfahren manuell, schrittweise durchgeführt und Arbeitsschritte und Zwischenergebnisse dokumentiert. Schließlich wurden die generierten Benutzerschnittstellen anhand von Screenshots vorgestellt. Im Nachgang zu der Generierung wurden die Benutzerschnittstellen den Experten (Therapeuten der Gelderlandklinik) vorgestellt, um zu bestimmen, ob der Interaktionsablauf und die Anpassung auf die Nutzungskontexte den definierten Anforderungen genügt. Von den Experten wurde bestätigt, dass die Ergebnisse der Generierung hinreichend gut sind und das Verfahren somit in dem geforderten Sinne die Zielsetzung erfüllt.

Aber auch Potentiale für Verbesserungen in den verwendeten Algorithmen konnten identifiziert werden. So schränkt die statische Zuordnung von BSF zu Layoutbereichen die Gestaltung der Benutzerschnittstelle stark ein. An den Screenshots erkennt man leicht, dass durch eine andere Positionierung der Elemente ein noch besseres Gesamtbild entstehen könnte. Auch könnten durch eine Verschmelzung von Bereichen oder eine Skalierung in ihrer Größe mehr Flexibilität erreicht werden. Weiterhin können die Algorithmen zur Sequenzialisierung, die in dieser Arbeit vorgestellt werden, verbessert werden. Die verwendete Methode setzt einen „Brute-Force“-Ansatz um, der vorgegebene Modifikationen ausführt und anschließend zu ermitteln versucht, ob die Manipulation erfolgreich war oder nicht. Es können aber BSF-AINs konstruiert werden, bei denen dieser Algorithmus auch eine Reihe von schlechten Lösungen produziert. Die manuelle Kontrollinstanz, welche die generierten Lösungen abschließend bewertet, stellt dabei sicher, dass nur geeignete Benutzerschnittstellen verwendet werden. Da dem Verfahren entsprechend nicht zur "Laufzeit" Benutzerschnittstellen generieren werden, sondern eine Generierung für gegebene Nutzungsprofile im Vorfeld erfolgt, relativieren sich somit auch Fragestellung der Effizienz- und Laufzeitbetrachtung der Algorithmen.

Weiterhin konnte im Rahmen der Evaluation gezeigt werden, dass bereits eine relativ kleine Wissensbasis, welche grundlegende Benutzerschnittstellenbausteine umfasst, ausreichend ist, um Ergebnisse zu erzeugen, die den im Nutzungskontext beschriebenen Anforderungen genügen. Schließlich wurde gezeigt, dass, wenn die notwendigen Wissensbasen bereitstehen, die Zielsetzung, nämlich die Generierung von Benutzerschnittstellen weitgehend zu automatisieren, erreicht werden konnte.

Die im Rahmen der Arbeit durchgeführte Evaluation kann naturgemäß keinen vollständigen Nachweis des erarbeiteten Verfahrens leisten. Ein formaler Beweis des Verfahrens ist aufgrund der Komplexität der domänenabhängigen Wissensbasen und der Problemstellung der Bewertung des Generierungsergebnisses (Benutzerschnittstelle geeignet oder nicht) nicht realistisch. Ein Aufzeigen der Funktionsfähigkeit und Sinnhaftigkeit des Verfahrens kann und sollte daher in einer breiten empirischen Untersuchung erfolgen. Die dazu erforderlichen Grundlagen sind mit den in dieser Arbeit durchgeführten Arbeitsschritten verfügbar.

Für die weitergehende Evaluation des Verfahrens wird aktuell am Fraunhofer-Institut für Software- und Systemtechnik ISST die Entwicklung einer geeigneten Werkzeugunterstützung zur Automatisierung des Verfahrens durchgeführt. Realisiert werden drei Bausteine: der "Comoda-Modeller" umfasst graphische Werkzeuge für die Erstellung der Wissensbasen (Interaktionsbeschreibung, OM-Katalog, Nutzungskontext); der "Comoda-Composer" realisiert die Automatisierung der Algorithmen; und die "Comoda-Execution" stellt die Laufzeitumgebung für die Benutzerschnittstellen bereit.

Parallel zur Bereitstellung der Werkzeugunterstützung erfolgt die Weiterentwicklung und Untersuchung des Anwendungsfalls "Adipositas Begleiter". Diese Evaluation wird im Rahmen des Forschungsprojektes "Realisierung und Evaluation eines Repositories für patientenorientierte telemedizinische Dienste" durchgeführt, das vom Land Nordrhein-Westfalen und der Europäischen Union im Rahmen des Wettbewerbs Med in.NRW unter dem **Förderkennzeichen 005-GW01-068** gefördert wird. Im Rahmen dieses Projektes erfolgt eine umfassende Analyse der Anwendungsdomäne und Modellierung der Wissensbasen, die für eine Evaluation des Verfahrens erforderlich sind. Zusammen mit Partnern aus dem fachlichen Gebieten (Therapeuten und telemedizinische Dienstleister) erfolgt die Bewertung der Generierungsergebnisse. Im Rahmen dieses Projektes werden darüber hinaus zwei weitere Anwendungsfelder aus dem Bereich der Telemedizin untersucht. Die Erfahrungen für die Umsetzung und der Vergleich der drei Anwendungsfeldern wird Daten zur Verfügung stellen, die es ermöglichen, Aussagen über die Güte der generierten Benutzerschnittstellen, den Aufwand für die Nachbearbeitung, über Laufzeitverhalten und Zeit- und Kostenersparnisse für die Realisierung zu geben.

10 Zusammenfassung und Resümee

Die Bereitstellung von Benutzerschnittstellen ist gegenwärtig ein vielschichtiges Aufgabenfeld. Es ist bei weitem nicht mehr ausreichend, Anwendungen auf unterschiedliche Endgeräte zu portieren und damit unterschiedliche Softwareplattformen zu bedienen. Endgeräte unterscheiden sich in einem viel größerem Umfang. Durch die Einführung von Multi-Touch-Displays, bewegungssensitiven Controllern (bspw. Nintendo Wii) und Gestensteuerung (durch 3D Videoaufnahmen) stehen neue Interaktionsformen zur Verfügung, die sich wesentlich vom Desktop-PC unterscheiden. Dabei bildet die Endgeräteproblematik nur einen Teil der eigentlichen Herausforderung. Benutzerschnittstellen müssen sich an den Bedürfnissen der Menschen orientieren, die sie schließlich nutzen sollen. Im Bereich AAL (Ambient Assisted Living) werden beispielsweise speziell IT-Systeme für ältere Menschen entwickelt. Der Desktop-PC nimmt im Bereich des AAL dabei nur eine untergeordnete Rolle ein. Gesucht sind hier Technologien, die sich in die Umgebung des Anwenders integrieren, die der ältere Mensch aber für die Interaktion mit IT-Systemen nutzen kann.

Wie diese Beispiele zeigen, ist in Anwendungsdomänen, die sich dadurch auszeichnen, dass sie heterogene Zielgruppen bedienen müssen, der Aufwand für die Benutzerschnittstellenentwicklung immens. So muss für jede Anwendung sichergestellt werden, dass für mögliche Endgeräte eine Portierung vorhanden ist. Notwendig ist es auch zu berücksichtigen, welche Möglichkeiten des Endgerätes der Anwender tatsächlich nutzen kann. Aber auch die Anwendung selbst muss den Erwartungen der Anwender entsprechen. Erfahrene Anwender werden sich eher Flexibilität in den Abläufen wünschen, während ältere Menschen lieber schrittweise geführt werden wollen. Will man diesen Anforderungen gerecht werden, müssen eine Menge von unterschiedlichen Benutzerschnittstellen für dieselbe Anwendung erstellt werden. In den wenigsten Fällen lässt sich der Aufwand für die manuelle Erstellung dieser Benutzerschnittstellen tatsächlich finanzieren. Hier sind Werkzeuge notwendig, welche eine automatisierte Erstellung von Benutzerschnittstellen abhängig von solchen Anforderungen ermöglichen.

Die vorliegende Arbeit versucht die Fragestellung zu beantworten, wie Benutzerschnittstellen für unterschiedliche Endgeräte, Anwender und auch Umgebungen der Nutzung effizient erstellt werden können. Als Anwendungsdomäne, an der diese Frage konkretisiert wird, wurden „Patientenorientierte telemedizinische Dienste“ und hier der Anwendungsfall „Adipositas-Begleiter“ ausgewählt (vgl. Kapitel 3). Der gewählte Ansatz sieht vor, die Generierung von Benutzerschnittstellen weitgehend automatisiert ablaufen zu lassen. Voraussetzung dafür ist, dass Modelle zur Verfügung stehen, mit denen beschrieben wird, welchen Anforderungen die zu generierende Benutzerschnittstelle gerecht werden muss und was die Benutzerschnittstelle leisten soll.

Ein Vergleich aktueller, modellbasierter Systeme zur Benutzerschnittstellengenerierung (vgl. Abschnitt 2) hat gezeigt, dass der Forschungsbereich viele innovative Lösungen bietet, die bereits eine hohe Reife erfahren haben. Obwohl auch in der Wirtschaft zunehmend erkannt wird, dass neue Methoden für die Unterstützung der Benutzerschnittstellengenerierung notwendig sind, konnten sich modellbasierte Systeme in der Praxis bisher aber nicht etablieren. Große Hersteller kommerzieller Software beginnen erst

formale Benutzerschnittstellenspezifikationen zu verwenden und entwickeln Laufzeitumgebungen für unterschiedliche Plattformen und Endgeräte, die in der Lage sind, diese auszuführen (vgl. Abschnitt 2).

Der in dieser Arbeit verfolgte Ansatz versucht nun beide Welten (modellbasierte Ansätze und entwicklungsnahe Spezifikationsmethoden) näher zusammen zu bringen. Die Idee, die dabei verfolgt wird, sieht vor, dass entwicklungsnahe Spezifikationsmethoden benutzt werden, um Benutzerschnittstellenbausteine zu entwickeln. Aus diesen Bausteinen sollen ausführbare Benutzerschnittstellen durch Komposition generiert werden. Wie die Komposition durchzuführen ist und welche Bausteine für eine Komposition in Frage kommen, wird dabei mit Hilfe modellbasierter Ansätze spezifiziert.

Um diese Aufgabenstellung zu konkretisieren, wurde der „Compositional Modeling“-Ansatz (vgl. Absatz 4.1) für die Benutzerschnittstellengenerierung in der folgenden Form adaptiert:

Gegeben seien eine Interaktionsbeschreibung in Form eines Abstract Interaction Nets (definiert in Abschnitt 5.3), eine „Domain Theory“ (definiert in Abschnitt 7.3) bestehend aus wieder verwendbaren Benutzerschnittstellenfragmenten (definiert in Abschnitt 7.2.1) und ein Nutzungsprofil (definiert in Abschnitt 6.3).

Aufgabenstellung war es, basierend auf den Vorgaben der Interaktionsbeschreibung durch Komposition der Benutzerschnittstellenfragmente der „Domain Theory“ eine ausführbare Benutzerschnittstelle (definiert in Abschnitt 7.2.2.2) zu generieren, welche den durch das Nutzungsprofil definierten Anforderungen bestmöglich genügt.

Dieser Aufgabenstellung entsprechend erfolgten zwei zentrale Arbeitschritte. In einem ersten Schritt wurden die Modelle (Interaktionsbeschreibung, Nutzungsprofil und Benutzerschnittstellenfragmente) entwickelt, indem basierend auf existierenden Vorarbeiten eine Spezifikationsmethodik ausgewählt oder weiter entwickelt wurde. Veranschaulicht wurde die Modellierung am Beispiel des Anwendungsfalls „Adipositas-Begleiter“, an dem präsentiert wurde, wie das Modell praktisch angewendet werden kann. Im zweiten Schritt wurde schließlich ein Verfahren erarbeitet, mit dem Benutzerschnittstellen aus diesen Modellen heraus durch Komposition generiert werden. Auch dieses Verfahren wurde am Beispiel des „Adipositas-Begleiters“ erläutert.

In einem ersten Schritt (vgl. Kapitel 5) wurde ein Modell für die Interaktionsbeschreibung entwickelt. Die primäre Zielsetzung bei der Wahl einer geeigneten Interaktionsbeschreibung war es, eine Balance zwischen Abstraktionslevel und Nähe zur Benutzerschnittstelle herzustellen. Um dies zu gewährleisten, wurden Objektmetaphern eingeführt, welche die Interaktionen eines Benutzers auf Ebene einer Benutzerschnittstelle beschreiben, ohne aber Aussagen über das Endgerät, den Nutzer und die Umgebung der Nutzung treffen zu müssen. Als Ausgangslage für die Spezifikation wurden „Dialognetze“ ausgewählt, die basierend auf Petri-Netzen ursprünglich für die Spezifikation fensterbasierter Benutzerschnittstellen eingesetzt wurden. Die Spezifikation der „Dialognetze“ wurde aufgrund der Anforderungen der Metaphern-orientierten Methodik zu den neu eingeführten AINs (Abstract Interaction Nets) weiterentwickelt. Weiterhin erfolgte die Spezifikation eines Objektmetaphern-Kataloges, mit dem Objekt- und Interakti-

onsmetaphern einer Domäne katalogisiert werden, um dem Modellierer ein Werkzeug zur Hand zu geben, mit dem er aus einer existierenden Wissensbasis heraus Benutzerschnittstellen beschreiben kann.

Nutzungsprofile (vgl. Kapitel 6) definieren im Rahmen des Generierungsprozesses Anforderungen an die zu generierende Benutzerschnittstelle. Für die Spezifikation solcher Anforderungen wurde CC/PP ausgewählt, da es Profile in einer Form beschreibt, die sich gut für eine maschinelle Weiterverarbeitung eignet. Weiterhin konnte gezeigt werden, dass eine manuelle Erstellung solcher Profile aufgrund des benötigten Umfangs für die Beschreibung problematisch ist. Um dieser Problemstellung zu begegnen, wurde ein Kontextmodell entwickelt, das die Modellierung von Nutzungskontexten ermöglicht, aus denen schließlich Nutzungsprofile automatisiert erzeugt werden können. Für das Anwendungsbeispiel „Adipositas-Begleiter“ wurde solch ein Nutzungskontextmodell aufgebaut, mit dem sich Nutzungskontexte modellieren lassen.

Benutzerschnittstellenbausteine (auch Benutzerschnittstellenfragmente genannt) wurden als „abgrenzbare und zusammengehörige Ausschnitte aus einem Interaktionsablauf“ beschrieben. Diese Definition erfolgte in Analogie zu den Objekt- und Interaktionsmetaphern, wie sie für die Interaktionsbeschreibung verwendet werden. Die Spezifikation für die Beschreibung von Benutzerschnittstellenfragmenten orientiert sich an den drei Teilbereichen einer Benutzerschnittstelle (Präsentation, Dialogsteuerung und Anwendungsschnittstelle): die Präsentation bedient sich einer XML-basierten Benutzerschnittstellenspezifikation, die Dialogsteuerung erfolgt auf Basis von Events und Event-Handlern, die Anwendungsschnittstelle wird schließlich über die Web Service Definition Language beschrieben.

Mit Hilfe der Beschreibung der Benutzerschnittstellenfragmente erfolgte die Definition der „Domain Theory“, die exemplarisch am Beispiel des „Adipositas-Begleiter“ instanziert wurde. Im Rahmen der Spezifikation der „Domain Theory“ wurde insbesondere der Nutzungskontextvektor für die Beschreibung von Eigenschaften von Benutzerschnittstellenfragmenten eingeführt. Dieser ermöglicht die Realisierung von Bewertungsfunktionen, die ermitteln können, wie gut ein Benutzerschnittstellenfragment für ein Nutzungsprofil geeignet ist.

Basierend auf den verfügbaren Modellen und Spezifikationen wurden Verfahren und Methoden zur Generierung von Benutzerschnittstellen entwickelt (vgl. Kapitel 8). Definiert wurde ein zweistufiges Verfahren. In einem ersten Schritt werden basierend auf den Anforderungen, die durch das Nutzungsprofil definiert sind, Benutzerschnittstellenfragmente identifiziert, die in der Lage sind, die Interaktionsbeschreibung zu realisieren. In einem zweiten Schritt werden die identifizierten Benutzerschnittstellenfragmente schließlich zu einer ausführbaren Benutzerschnittstelle komponiert. Zur Identifikation der Benutzerschnittstellenfragmente wird der OM-Katalog als primäre Wissensbasis genutzt. Anhand des OM-Kataloges werden mögliche Realisierungen (AINs oder BSF) von Interaktionsmetaphern der Interaktionsbeschreibung ermittelt und mit Hilfe einer Bewertungsfunktion gewichtet. Eingesetzt wird die Dekomposition, um Interaktionsmetaphern in Sub-Netze zu zerlegen, die Selektion aus Realisierungsalternativen (BSF, AINs) und die Komposition von Alternativen. Das Ergebnis des ersten Generierungsschrittes ist ein sogenanntes BSF-AIN, das sich der Syntax eines AINs bedient, für das aber Abbildungen auf Benutzerschnittstellenfragmente der „Domain Theory“ existieren. Im zweiten Generierungsschritt

werden die im BSF-AIN modellierten Beziehungen zwischen den Benutzerschnittstellenfragmenten untersucht und überprüft, ob sich diese auf das im Nutzungsprofil angegebene Layoutpattern abbilden lassen. Ist dies nicht der Fall, so werden Algorithmen zur Sequenzialisierung eingesetzt, um mögliche Konflikte aufzulösen. Diese Algorithmen manipulieren das BSF-AIN so lange, bis keine Konflikte mehr vorhanden sind. Erst dann erfolgt die Komposition der Benutzerschnittstellenfragmente mit Hilfe einer entwickelten Intervallalgebra, mit der die im BSF-AIN modellierten Beziehungen in Regeln für die Komposition der Benutzerschnittstellenfragmente überführt werden können.

Den Abschluss der Arbeit bildete die praktische Anwendung des entwickelten Verfahrens zur Generierung von Benutzerschnittstellen als „Proof of Concept“. Auf drei unterschiedliche Nutzungskontexte wurde das in der Arbeit entwickelte Verfahren angewendet und ausführbare Benutzerschnittstellen generiert. Dabei wurde das Verfahren manuell durchgeführt, und Arbeitsschritte und Zwischenergebnisse wurden dokumentiert. Schließlich wurden die generierten Benutzerschnittstellen anhand von Screenshots vorgestellt und Problemstellungen und Schwächen des Ansatzes beleuchtet.

Die Evaluation konnte zeigen, dass die zentrale Zielsetzung, nämlich die automatisierte Generierung auf Basis des „Compositional Modeling“-Ansatzes von Benutzerschnittstellen für unterschiedliche Nutzer, Endgeräte und Umgebungen erreicht werden konnte. Damit konnte der Nachweis erbracht werden, dass das in der Arbeit entwickelte Verfahren geeignet ist, um Ansätze für eine modellbasierte-Beschreibung von Benutzerschnittstellen mit der Komposition von implementierungsnahen Spezifikationen zu verbinden.

Damit leistet die vorliegende Arbeit auch einen Beitrag für einen „Brückenschlag“ von der Forschung in die Praxis. Während modellbasierte Verfahren zur Benutzerschnittstellengenerierung zur Zeit primär im Bereich der Forschung diskutiert und entwickelt werden, finden sich in der Praxis meist nur implementierungsnahe Spezifikationsmethoden (bspw. XAML) und Werkzeuge zur Erstellung solcher Spezifikationen (GUI-Builder).

Zwei zentrale Leistungen der Arbeit sind im Rahmen des entwickelten Verfahrens besonders hervor zu heben. Einerseits ist dies die Metaphern-orientierte Spezifikation der Interaktionsbeschreibung, die Benutzerschnittstellen auf einer Interaktionsebene beschreibt, und dadurch eine neue Abstraktionsebene zwischen Task-Modellen und der konkreten Spezifikation der Benutzerschnittstelle schafft. Damit fungiert die Interaktionsbeschreibung als Brücke zwischen Modell-basierten Ansätzen und implementierungsnahen Spezifikationen. Die Verwendung von Metaphern liefert weiterhin einen interessanten Ansatz, formale Spezifikationen mit der semantischen Beschreibung von Objekten zu verbinden.

Eine weitere Leistung dieser Arbeit ist die **Ausrichtung** des entwickelten Verfahrens in Richtung eines Baukastens für die Generierung von Benutzerschnittstellen. Diese Bausteinidee reduziert sich dabei nicht nur auf die „Domain Theory“, sondern findet sich auch im OM-Katalog (Baukasten für die Interaktionsbeschreibung) und auch im Kontextmodell (Baukasten für Nutzungsprofile) wieder.

Mit der Entwicklung des Verfahrens, der Konzeption von Modellen, Wissensbasen und Methoden zur automatisierten Generierung und schließlich der Evaluation anhand des Anwendungsbeispiels „Adipositas-Begleiter“ ist die Aufgabenstellung der Arbeit abgeschlossen.

Die Bereitstellung dieser Ergebnisse ermöglicht nun auch die Kommunikation der Leistungen in die wissenschaftliche Community. Im Laufe des Dissertationsvorhabens wurden umfangreichen Arbeiten für die Konzeption von Modellen, Wissensbasen, und Algorithmen geleistet, die einzeln den aktuellen Stand der Wissenschaft nur punktuell erweitern. Bevor eine Veröffentlichung der Ergebnisse möglich war, musste erst die Eignung der einzelnen Lösungsbausteine (Modelle, Wissensbasen und Algorithmen) im Rahmen eines Verfahrens nachgewiesen werden. Erst die Evaluierung im Rahmen der vorliegenden Arbeit zeigt, dass ein Verfahren angegebenen werden kann, das unter Nutzung der einzelnen Lösungsbausteine in der Lage ist, Benutzerschnittstellen zu generieren.

Die Ergebnisse der vorliegenden Arbeiten lassen auch Raum für Weiterentwicklungen und Verbesserungen des Verfahrens. So wurde die verwendete Wissensbasis (OM-Katalog, Kontextmodell, „Domain Theory“) nur rudimentär für den Anwendungsfall des „Adipositas-Begleiter“ erstellt. Es fehlt eine weitergehende Evaluation dieser Wissensbasen, die bisher nur anhand praktischer Erfahrungen in einem Anwendungsfall erstellt wurden. Wichtig ist es hier zu hinterfragen, ob beispielsweise die angegebenen Nutzungskontextelemente tatsächlich diejenigen sind, die für die Generierung relevante Parameter definieren. Schließlich wird auch eine Evaluation benötigt, wie sich diese Parameter auf die Benutzerschnittstelle auswirken müssen. Dies gilt ebenso für die Defaultprofile, aber auch der aktuell verfügbare OM-Katalog bildet nur einen Ausschnitt aus der Anwendungsdomäne ab.

Die Weiterführung der hier vorgestellten Arbeiten erfolgt im Rahmen des Med in.NRW Projektes "Realisierung und Evaluation eines Repositories für patientenorientierte telemedizinische Dienste" und ermöglichen eine weitergehende Bewertung des erarbeiteten Verfahrens.

Die skizzierten Arbeiten definieren neue Aufgabenfelder, die nicht mehr im Fokus der vorliegenden Arbeit liegen. Zielsetzung war die Konzeption und Evaluation eines bausteinorientierten Verfahrens für die Generierung von Benutzerschnittstellen, wobei auch die praktische Anwendbarkeit gezeigt werden sollte. Dass im Rahmen dieses Verfahrens Effizienzsteigerungen und auch Verbesserung zu erzielen sind, indem identifizierte Module optimiert und definierte Wissensbasen (Bibliotheken und Kataloge) weiter ausgebaut werden, war zu erwarten. Die Möglichkeiten für weitergehende Optimierungen der einzelnen Bestandteile des Verfahrens für die komplexe Problemstellung der Benutzerschnittstellengenerierung unterstreichen das Potential des entwickelten Verfahrens.

Literaturverzeichnis

[AB91] G. D. Abowd, R. Beale: **Users, systems and interfaces: A unifying framework for interaction.** HCI'91: People and Computers VI, Cambridge University Press, Cambridge, 1991, S.73-87

[ACM92] ACM SIGCHI Curriculum Development Group: **ACM SIGCHI Curricula for Human Computer Interaction**. ACM, New York, 1992

[All83] J. F. Allen: **Maintaining Knowledge about Temporal Intervals.** Communications of the ACM, 1983, Vol.26, No.11, S. 832-843

[APA04] M. F. Ali , M. A. Pérez-Quiñones, M. Abrams: **Building Multi-Platform User Interfaces with UIML.** Multiple User Interfaces: Engineering and Application Framework, John Wiley & Sons, Chichester, 2004, S. 95-118

[Bal05] H. Balzert: **Lehrbuch der Objektmodellierung, Analyse und Entwurf mit der UML 2**, Spektrum Akademischer Verlag, Heidelberg, 2005

[BB04] D. Beckett, B. McBride: **RDF/XML Syntax Specification (Revised)**, W3C Recommendation 10 February 2004, 2004, http://www.w3.org/TR/REC-rdf-syntax/ (zuletzt besucht am 16.05.2010)

[BCFM07] M. Brambilla, S. Comai, P. Fraternali, M. Matera: **Designing Web Applications with WebML and WebRatio**, Hrs. G. Rossi, Web Engineering: Modelling and Implementing Web Applications, Springer Verlag, Berlin, 2007

[BCP04] S. Berti, F. Correani, F. Paternò, C. Santoro: **The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Leveles**, Proceedings Workshop on Developing User Interfaces with XML UIXML: Advances on User Interface Description Language, 2004, S. 103-110

[BFJ96] H-J. Bullinger, K-P. Fähnrich, C. Janssen: **Ein Beschreibungskonzept für Dialogabläufe bei graphischen Benutzerschnittstellen**, Informatik Forsch. Entw., 1996, S. 84-93

[BG04] M. Book, V. Gruhn: **Modelling Web-Based Dialog Flows for Automatic Dialog Control**, 19th IEEE International Conference on Automated Software Engineering, 20-25 September 2004, IEEE Computer Society, Linz, Austria, 2004 S. 100-109

[BGG02] BGG: **Behindertengleichstellungsgesetz vom 01. Mai 2002**, in: BGBl. I/2002, 2002

[BHS08] W. Beinhauer, M. Herr, A. Schmidt: **SOA für Agile Unternehmen**, Symposion Publishing GmbH, Düsseldorf, 2008

[BIT09] BITCOM, Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V.,: **Internet-Fernsehen boomt**, BITCOM-Presseinformation, Berlin, 17. Juni 2009, http://www.bitkom.org/files/documents/BITKOM-Presseinfo_IPTV_17_06_2009.pdf
(zuletzt besucht am 16.05.2010)

[BITV02] BITV: **Barrierefreie Informationstechnikverordnung vom 17. Juli 2002**, in: BGBl. I/2002, 2002

[BLC03] J. Van Den Bergh, K. Luyten, K. Coninx: **A Run-time System for Context-Aware Multi-Device User Interfaces**, HCI International 2003, 2003, S. 308-312

[Bus45] V. Bush: **“As We May Think"**, The Atlantic Monthly, (Reprinted and discussed in interactions, 3, 1996), 1945

[CCN97] G. Calvary, J. Coutaz, L. Nigay: **From Single-User Architectural Design to PAC*:a Generic Software Architecture Model for CSCW**, Proceedings of ACM CHI 97, 1997, S. 242-249

[CG00] H. Christensen, K. Griffiths: **The Internet and Mental Health Literacy**, Australian & New Zealand Journal of Psychiatry, 2000, S. 975-979

[CH03] S. Crowle, L. Hole: **ISML: An Interface Specification Meta-language**, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2003, S. 255-268

[CK00] G. Chen, D. Kotz: **A survey of context-aware mobile computing research.** Technical Report TR2000-381, Computer Science Department, Dartmouth College, 2000

[CMN84] S. K. Card, T. P. Moran, A. Newell: **The psychology of human-computer interaction**, Lawrence Erlbaum Associates, New Jersey, 1984

[CSS2] Cascading Style Sheets level 2 Revision 1 (CSS 2.1): **CSS2 Spezification,** W3C, 2009

[CVC08] B. Collignon, J. Vanderdonckt, G. Calvary: **Model-Driven Engineering of Mulit-target Plastic User Interfaces**, ICAS 08: Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems, IEEE Computer Society, Washington, DC, USA, 2008, S. 7-14

[Das01] P. P. Da Silva: **User interface declarative models and development environments: A survey**, lecture notes in computer science; Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, 2001, S. 207-226

[DD08] B. Dayley, L.D.N. Dayley: **Silverlight 2 Bible**, John Wiley & Sons, Inc., New Jersey, USA, 2008

[Den91] E. Denert: **Software Engineering - Methodische Projektabwicklung**, Springer Verlag, Berlin, 1991

[Dey01] A. Dey: **Understanding and Using Context**, Personal and Ubiquitous Computing Journal, 2001, Vo.5 No.1, S. 4-7

[DKSS09] T. Döring, A. Krüger, A. Schmidt, J. Schöning: **Tangitable, Emedded, and Reality-Based Interaction**, it - Information Technology, 2009, 6, S. 319-324

[DL01] Hrsg.: W. Deiters, C. Lienemann: **Report Informationslogistik**. Symposion Publishing, Düsseldorf, 2001

[Dra05] J. Draeger: **Evaluation XML-basierter Beschreibungssprachen für Multiplattform-Benutzungsschnittstellen**, Diplomarbeit an der TU-Dortmund, 2005

[DUDEN82] Wissenschaftlicher Rat d. Dudenredaktion: Hrsg.: D. Drosdowski: **Der Duden in 10 Bänden, Duden „Fremdwörterbuch“**. Band 5. 4. Auflage, Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, 1982, S. 350

[ECMAS] ECMAScript: **Standard ECMA-262**, ECMAScript Language Specification, 3rd edition, ECMA, 1999, http://www.ecma-international.org/publications/standards/Ecma-262.htm, (zuletzt besucht am 16.05.2010)

[EVP01] J. Eisenstein, J. Vanderdonckt, A. R. Puerta: **Applying model-based techniques to the development of UIs for mobile computers**, Proceedings of the 6th international conference on Intelligent user interfaces, 2001, S. 69-76

[FCJ*08] F. Paterno, C. Santoro, J. Mantyjarvi, G. Mori, S. Sansone: **Authoring pervasive multimodal user interfaces**, International Jounal of Web Engineering and Technology, 2008, Vol. 19, S. 695-720

[FDR*04] P. Forbrig, A. Dittmar, D. Reichart, D. Sinnig: **From Models to Interactive Systems Tool Support and XIML**. Hrsg.: H. Trætteberg, P. J. Molina, N. J. Nunes : Proceedings of the First International Workshop on Making model-based user interface design practical: usable and open methods and tools, Funchal, Madeira, Portugal. CEUR-WS.org, 2004

[Fen03] X. Feng : **Konzeptuelle Metaphern und Textkohärenz**, Gunter Narr Verlag, Tübingen, 2003

[FF91] B. Falkenhainer, K. D. Forbus: **Compositional modeling: finding the right model for the job**, Artificial Intelligence, 1991, Vol. 51, S. 95-143

[FHSS09] M. Feldmann, G. Hübsch, T. Springer, . Schill: **Improving Task-driven Software Development Approaches for Creating Service-Based Interactive Applications by Using Annotated Web Services**, 2009 Fifth International Conference on Next Generation Web Services Practices, 2009, S. 94-97

[FL03] P. Fettke, P. Loos: **Entwicklung eines Metamodells für die "Vereinheitlichte Spezifikation von Fachkomponenten**, Modellierung und Spezifikation von Fachkomponenten;4. Workshop im Rahmen der Modellierung betrieblicher Informationssysteme (MobIS), Bamberg, 2003, S. 13-21

[For02] P. Forbrig: **Objektorientierte Softwareentwicklung mit UML**, Fachbuchverlag, Leipzig, 2002

[FSZ07] X. Fu, Q. Shen, R. Zhao: **Towards fuzzy Compositional Modeling**, In Proceedings of the 16th IEEE, International Conference on Fuzzy Systems, 2007, S. 1233-1238

[GC10] K. Gaßner, M. Conrad, **ICT enabled independent living for the elderly, A status-quo analysis on products and the research landscape in the field of Ambient Assisted Living (AAL) in EU-27**, Innovation und Technik, VDI, 2010

[GCT*03] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J. Vanderdonckt: **A unifying reference framework for multi-target user interfaces**, Journal of Interacting With Computer;Elsevier Science B.V, 2003, Vol.15, No.3, S. 289-308

[GCVA09] J. Guerrero Garcia, J. M. Ganzalez-Calleros, J. Vanderdonckt, J. Munoz Arteaga: **A Theoretical Survey of User Interface Description Languages: Preliminary Results**, LA-WEB/CLIHC, 2009, S. 36-43

[GM91] M. Green, R. Jocob: **SIGGRAPH 90 Workshop report: software architectures and metaphors for non-WIMP user interfaces**, ACM SIGGRAPGH Computer Graphics, ACM, New York, USA, 1991, S. 229-235

[GMF*98] T. Griffiths, J. McKirdy, G. Forrester, N. Paton, J.´Kennedy, P. Barclay, R. Cooper, C. Goble, P. Gray: **Exploiting Model-Based Techniques for User Interfaces to Database**, IFIP Conference Proceedings; 1998, Vol. 126, S. 21-46

[GMP*98] T. Griffiths, J. McKirdy, N. Paton, J. Kennedy, R. Cooper, P. Barclay, C. Goble, P. Gray, M. Smyth, A. Dinn: **An Open Model-Based Interface Development System: The Teallach Approach**, Proceedings of DS-VIS'98, 1998, S. 32-49

[Göt94] R. Götze: **Dialogmodellierung für multimediale Benutzerschnittstellen**, Universität Oldenburg, Fachbereich Informatik, Dissertation, 1994

[GPR06] V. Gruhn, D. Pieper, C. Röttgers: **MDA-Effektives Software-Engineering mit UML2 und Eclipse**, Springer Verlag, Berlin, S. 2006

[Gre86] M. Green: **Survey of Three Dialogue Models**, ACM Transactions on Graphics, 1986. Vol. 5, S. 247-275

[Grub95] T. R. Gruber: **Towards principles for the design of ontologies used for knowledge sharing**, International Journal of Human-Computer Studies, 1995, Vol. 43, S. 907-928

[HA08] J. Helms, M. Abrams: **Retrospective on UI description languages, based on eight years experience with the User Interface Markup Language (UIML)**, International Journal of Web Engineering and Technology, 2008, Vol. 2, S. 138-162

[Hän04] S. Hänke: **Anthropomorphisierende Metaphern in der Computerterminologie - eine korpusbasierte Untersuchung**, M.A. thesis at Humboldt-University, Berlin, 2004

[Har88] D. Harel: **On Visual Formalism of Complex Systems**, Communication of the ACM, 1988, Vol. 31, No. 5, S. 514-530

[Has04] S. Haseloff: **Context Gathering – an Enabler for Information Logistics**, Multikonfe-renz Wirtschaftsinformatik (MKWI) 2004, 2004, Vol.2, S. 204-216

[HP91] P. V. Hentenryck, T. L. Provost: **Incremental Search in Constraint Logic Programming**. New Generation Computing, Ohmsha, Ltd., 1991, Vol. 9, No. 3-4, S. 257-275

[HS98] U. Heller, P. Struss: **Modellbasierte Entscheidungsunterstützung im Umweltbereich mit qualitativen Modellen**, KI, 1998, Vol.12, No.2, S. 35-38

[HSJ09] J. Hong, E. Suh, S. Kim: **Context-aware systems: A literature review and classification**, Expert Systems with Applications: An International Journal, 2009, Vol. 36, S. 8509-8522

[HSLV*08] J. Helms, R. Schaefer, K. Luyten, J. Vanderdonckt, J. Vermeulen, M. Abrams: **User Interface Markup Language (UIML) Version 4.0,** Committee Draft, OASIS User Interface Markup Language (UIML) TC, 2008

[IRR*03] J. Indulska, R. Robinson, A. Rakotonirainy, K. Henricksen: **Experiences in Using CC/PP in Context-Aware Systems**, Mobile Data Management 2003, 2003, S. 247-261

[Jäc09] Telemedizinführer 2009: Hrs. A. Jäckel, **Telemedizinführer Deutschland**, Minerva KG Gude, Darmstadt, 2008

[Jan93] C. Janssen: **Dialognetze zur Beschreibung von Dialogabläufen in graphisch-interaktiven Systemen**, Software-Ergonomie,1993, S. 67-76

[JK06] F. Joualt, I. Kurtev: **Transformating Models with ALT**, Lecture Notes in Computer Science, Berlin, Springer Verlag, Berlin, 2006, S. 128-138

[JK96] B. E. John, D. E. Kieras: The GOMS family of user interface analysis techniques: comparison and contrast, ACM Transactions on Computer-Human Interaction, 1996, Vol. 3, S. 320-351

[Joh05] M. Jonathan: **Views 2: Reflections on Views**, MSc Thesis, University of Victoria, Canada, 2005

[KBN*04] R. Khaled, P. Barr, J. Noble, R. Biddle: **System Metaphor in Extreme Programming: A Semiotic Approach**, Proceedings of the 7th International Workshop on Organisational Semiotics, 2004

[Kiss07] C. Kiss: **Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 2.0**, W3C Working Draft 30 April 2007, 2007, http://www.w3.org/TR/CCPP-struct-vocab2/ (zuletzt besucht am 16.05.2010)

[KK08] T. Königsmann, R. Kriebel: **Digitale Gesundheitsbegleiter am Beispiel der Adipositas-Nachsorge**, 1. Deutscher AAL Kongress. Technologien - Anwendungen – Management, VDE-Verlag, Berlin, 2008, S. 5

[KKK07] A. Kraus, A. Knapp, N. Koch: **Model-Driven Generation of Web Applications UWE**, In Proc. MDWE 2007 - 3rd International Workshop on Model-Driven Web Engineering (CEUR-WS), 2007

[KKZB08] N. Koch, A. Knapp, G. Zhang, H. Baumeister: **Uml-Based Web Engineering: An Approach Based on Standards**, Hrsg. G. Rossi, O. Pastor, D. Schwabe, L. Olsina, Web Engineering: Modelling and Implementating Web Applications, Springer Verlag, London, 2008

[KLWK06] T. Königsmann, F. Lindert, R. Walter, R. Kriebel: **Hilfe zur Selbsthilfe als Konzept für einen Adipositas-Begleiter**, HMD - Praxis Wirtschaftsinform, 2006, Vol. 251, S. 64-76

[Koc07] N. Koch: **Classificatoin of Model Transformation Techniques used in UML-based Web Engineering**, IET Software Jounal (Institution of Engineering and Technology), 2007, Vol.1 No.3, S. 98-111

[Kuss05] T. Kussmaul: **Die Java Portlet Spezifikation**, JAVASPEKTRUM, Vol. 3, 2005

[Lan97] P. Langley: **Machine Learning for Adaptive User Interfaces**, Proceedings of the 21st German Annual Conference on AI, 1997, S. 53-62

[LCA05] K. Luyten, K. Coninx , M. Abrams: **Integrating UIML, task, dialogs with layout patterns for multi-device user interface design**, In: Proceedings of HCI International 2005, 2005, S. 22-27

[Lew00] J. Lewerenz: **Automatic Generation of Human-Computer Interaction in Heterogeneous and Dynamic Environments Based on Conceptual Modelling**, Brandenburgische Technische Universität, Fakultät für Mathematik, Naturwissenschaften und Informatik, Cottbus, 2000

[LJ80] G. Lakoff, M. Johnson: **Metaphors We Live By**, University of Chicago Press, Chicago, 1980

[LLPR*09] M. Linaje Trigueros, A. Lozano Tello, J. C. Preciado Rodríguez, R. Rodríguez Echeverría, F. Sánchez Figueroa: **Obtaining accessible RIA UIs by combining RUX-Method and SAW**, Fifth International Workshop, Automated Specificatoin and Verification of Web Systems, 2009, S. 85-99

[LRSV*09] J. Helms, K. Luyten, R. Schaefer, J. Vermeulen, M. Abrams, A. Coyette, J. Vanderdonckt, **Human-Centered Engineering of Interactive Systems with the User Interface Markup Language**, Hrsg. A. Seffah, J. Vanderdonckt, M. Desarais, Human-Centered Software Engineering, Springer Human-Computer Interaction Series, Springer Verlag, Berlin, 2009, S. 139-171

[LVM*04] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, M. Florins, D. Trevisan: **UsiXML: A user interface description language for context-sensitive user interfaces**. In International Working Conference on Advanced Visual Interfaces (ACM AVI 2004) Workshop: UIXML 2004, 2004, S. 55-62

[LVM*05] Q. Limbourg, J. Vanderdonckt, B. Michotte, L.Bouillon, V. Lopez: **UsiXML: a Language Supporting Multi-Path Development of User Interfaces**, in Proc. of 9th IFIP Working Conf. on Engineer-ing for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS 2004; Lecture Notes in Computer Science, Springer Verlag, Berlin, 2005

[LVN00] Q. Limbourg, J. Vanderdonckt, N. Souchon: **The Task-Dialog and Task-Presentation Mapping Problem: Some Preliminary Results**, In Proceedings of DSV-IS'2000, 2000, S. 227-246

[Mad94] K. H. Madsen: **A Guide to Metaphorical Design**, Communications of the ACM, 1994, Vol. 37, S. 57-62

[MFC01] A. Müller, P. Forbrig, C.H. Cap: **Model-Based User Interface Design Using Markup Concepts**, Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification-Revised Papers, 2001, S. 16-27

[MHF00] B. A. Myers, S. E. Hudson, F. R. Pausch: **Past, present, and future of user interface software tools**, ACM Transactions on Computer-Human Interaction, 2000, Vol. 7, S. 3-28

[MJPL92] S. Minton, M. D. Johnston, A. B. Philips, P. Laird: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, 1992, Vol. 58, No. 1-3, S. 161-205

[ML06] F. Montero, V. Lopez-Jaquero: **IdealXML: An Interaction Design Tool-A Task-Based Approach to User Interface Design**, Proc. of 6th. Int. Conf. on Computer-Aided Design of User Interfaces, Springer Verlag, Berlin, 2006, S. 245-252

[Moz10] Mozilla Foundation: **Fennec**, https://developer.mozilla.org/En/Mobile (zuletzt besucht am 16.05.2010)

[MPS02] G. Mori, F. Paterno, C. Santono: **An XML Based Approach for Designing Nomadic Applications**, Proc. W3C Device Independent Authoring Techniques Workshop, 2002

[MPS04] G. Mori, F. Paternò, C. Santoro: **Design and Development of Multidevice User Interfaces through MultipleLogical Descriptions**, IEEE Transactions on Software Engineering, 2004, S. 507-520

[MS09] G. Meixner, R. Schäfer: **Modellbasierte Entwicklung von Benutzerschnittstellen mit UIML**, i-com: "Mensch-Computer-Interaktion im Operationssaal", 2009, Vol. 8, S. 60-67

[MV08] B. Michotte, J. Vanderdonckt: **GrafiXML, A Multi-Target User Interface Builder based on UsiXML**, Proc. of 4th International Conference on Automatic and Autonomous System, IEEE Computer Society Press, Washington, DC, USA, 2008

[Mye95] B. A. Myers: **User Interface Software Tools**, Transactions on Computer-Human Interaction, 1995, Vol.2 No.1, S. 64-103

[Mye98] B. A. Myers, **A Brief History of Human Computer Interaction Technology**, ACM interactions, 1998, Vol. 5, No. 2, S. 44-54

[NC93] L. Nigay, J. Coutaz, **A Design Space for Mulit-Modal Systems: Concurrent Processing and Data Fusion**, Proceedings INTERCHI93, ACM/IFIP, 1993, S. 172-176

[NDW06] J. Neuhaus, W. Deiters, M. Wiedeler, **Mehrwertdienste im Umfeld der elektronischen Gesundheitskarte**, Informatik-Spektrum, 2006, Vol. 29, S. 332-340

[Nor86] D. Norman: **Cognitive Engineering**. Hrsg.: D. Norman, S. Draper: User-Centered System Design. Erlbaum, Hillsdale NJ., 1986

[Oes05] B. Oestereich: **Analyse und Design mit UML 2**, Oldenbourg Verlag, München, 2005

[Ol92] D. R. Olsen: **User Interface Management System: Models and Algorithms**, Morgan Kaufmann, San Mateo, CA, 1992

[OMA06] Open Mobile Alliance: **User Agent Profile (UAProf) version 2.0**, OMA specification, 2006, http://www.openmobilealliance.org/technical/release_program/uap_v2_0.aspx (zuletzt besucht am 16.05.2010)

[OMG04] Object Management Group Inc.: **UML 2.0 Infrastructure Final Adopted Specifcation**, 2004, http://www.omg.org/cgi-bin/doc?ptc/2003-09-15 (zuletzt besucht am 16.05.2010)

[OO91] J.R. Olson, G. M. Olson: **The growth of cognitive modeling in human computer interaction since GOMS**, Human Computer Interaction, 1991, Vol. 6, S. 21-30

[ORK97] R. Oppermann, R. Rashev, K. Kinshuk: **Adaptability and adaptivity in learning systems**, Knowledge Transfer. Vol.2 : Proceedings on Knowledge Transfer, London, 1997, S. 173–179

[Pat01] F. Paternò: **Task Models in Interactive Software Systems**. Hrsg.: S. K. Chang: Handbook of Software Engineering & Knowledge Engineering. World Scientific Publishing Co., River Edge, NJ, USA, 2001

[Pat99] F. Paternò: **Model-based Design and Evaluation of Interactive Applications**, Springer-Verlag, London, 1999

[PE02] A. Puerta ; J. Eisenstein: **XIML: A Common Representation for Interaction Data**, Proceedings of the 7th international conference on Intelligent user interfaces, 2002, S. 214 - 215

[Pel03] C. Peltz: **Web Services Orchestration and Choreography**, IEEE Computer, 2003, Vol.36 No.10, S. 46-52

[Pha00] C. Phanouriou: **UIML: A Device-Independent User Interface Markup Language**, PhD thesis, Vermount University, 2000

[PLSZ*08] J. C. Preciado, M. Linaje, F. Sanchez-Figueroa, G. Zhang, C. Kroiß, N. Koch: **Designing Rich Internet Applications Combining UWE and RUX-Method**, In Proc. of International Conference on Web Engineering (ICWE 2008), IEEE Computer Society, Washington, DC USA, 2008

[PMM97] F. Paternò, C. Mancini, S. Meniconi: **ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models**, IFIP Conference Proceedings; Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction, 1997, Vol. 96, S. 362-369

[Pri96] C. Pribeanu: **Task Modeling for User Interface Design: A Layered Approach**, International Journal of Information Technology, 2006, Vol. 3, No. 2

[PS03] F. Paternò, C. Santoro: **A Unified Method for Designing Interactive Systems Adaptable To Mobile And Stationary Platforms**, Interacting with Computers, Elsevier, 2003, Vol. 5, No.3, S. 347-364

[PSMM*08] F. Paterno, C. Santoro, J. Mantyjarvi, G. Mori, S. Sandone, **Authoring pervasive multimodal user interfaces**, International Journal of Web Engineering and Technology 2008, Vol. 4, S. 235-261

[PSS09] F. Paterno, C. Santoro, L.C. Spano: **MARIA: A universal, declarative, multi abstraction-level language for service-oriented application in ubiquitous environments**, ACM Transaction on Computer-Human Interaction, New York, USA, ACM, 2009

[Pue97] A.R. Puerta: **A model-based interface development environment**, IEEE Software, IEEE Computer Society Press, Los Alamitos, CA, USA, 1997, Vol. 14, No 4, S. 40-47

[RB01] E. Rahm, P. A. Bernstein: **A survey of approaches to automatic schema matching**, VLDB Journal, 2001, Vol.10, No.4, S. 334–350

[RKKR*09] D. Ruiz-Gonzalez, N. Koch, C. Kroiss, J. Romero, A. Vallecillo: **Viewpoint Synchronization of UWE Modells**, Proc. MDWE 2009, 5rd International Workshop on Model-Driven Web Engineering, CEUR-WS, 2009

[Roz97] G. Rozenberg: **Handbook of Graph Grammars and Computing by Graph Tranformation**, World Scientific, Singapore, 1997

[RS06] G. Rossi, D. Schwabe: **Model Based Web Application**, Hrsg. E. Mendes, N. Mosley, Web Engineering, Springer Verlag, Berlin, 2006, S. 303-333

[Rum88] J. Rumbaugh: **State Trees as Structured Finite State Machines for User Interfaces**, Proceedings of the ACM Symposium on User Interface Software, 1988, S. 15-29

[SAW94] B. Schilit, N. Adams, R. Want: **Context-Aware Computing Applications**, 1st International Workshop on Mobile Computing Systems and Applications, 1994, S. 85-90

[SBG99] A. Schmidt, M. Beigl, H.-W. Gellersen, **There is more to context than location**, Computers and Graphics, 1999, Vol.23, No.6, S. 893-901

[Sch01] A.-W. Scheer: **ARIS – Modellierungsmethoden, Metamodelle, Anwendungen**, Springer Verlag, Berlin, 2001

[Sch94] S. Schreiber: **Specification and Generation of User Interfaces with the BOSS-System**. Lecture Notes in Computer Science; Human Computer Interaction, Selected Papers EWHCI'94 Conference, Springer Verlag, München, 1994, S.107-120

[SFG93] P. Sukaviriya, J. D. Foley, T. Griffith: **A Second Generation User Interface Design Environment: The Model and The Rundimte Architecture**, Proceedings INTERCHI93: Human Factors in Computing Systems, Amsterdam, 1993, S. 375-382

[SJGL08] Shaer, R. Jocob, M. Green, K. Luyten, **User interface description languages for next generation user interfaces**, CHI 08 extended abstract on Human factors in computing systems, ACM, New York, USA, 2008, S. 3949-3952

[SS97] D. Sackett, J. Snow: **Compliance in Health Care**, Hrsg. R. Haynes, D. Taylor, D. Sacket, The magnitude of compliance and non-compliance, Johns Hopkins University Press, Baltimore, 1997, S. 11-22

[ST04] K. Samaan, F. Tarpin-Bernard: **Task models and interaction models in a multiple user interfaces generation process,** ACM International Conference Proceeding Series; Proceedings of the 3rd annual conference on Task models and diagrams, 2004, Vol. 86, S. 137-144

[ST94] B. Schilit, M. Theimer: **Disseminating Active Map Information to Mobile Hosts**, IEEE Network, 1994, Vol.8 No.5, S. 22-32

[Str96] S. Strahringer: **Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysemethoden**, Shaker Verlag GmbH, Aachen, 1996

[Suc96] B. Sucrow: **Formal Specification of Graphical User Interfaces, Using Graph Grammars**, Software Engineering im Scientific Computing, Vieweg Verlag, Wiesbaden, 1996, S. 279-289

[SV03] N. Souchon, J. Vanderdonckt: **A Review of XML-compliant User Interface Description Languages**, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2003, S. 377-391

[SV93] T. Schiex, , G. Verfaillie: **Nogood Recording for Static and Dynamic Constraint Satisfaction Problems**, International Journal of Artificial Intelligence Tools, 1993, S. 187-207

[SW01] R. Salfeld, J. Wettke: **Die Zukunft des deutschen Gesundheitswesens**, Springer-Verlag, Berlin, 2001

[Sze96] P. Szekely: **Retrospective and Challenges for Model-Based Interface Development**, Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces, Namur University Press, 1996

[TC99] D. Thvenin, J. Coutaz: **Adaptation and Plasticity of User Interfaces**, i3-spring99 Workshop on Adaptive Design of Interactive Multimedia Presentations for Mobile Users, 1999

[TNS09] TNS Infratest Holding GmbH & Co. KG, Initiative D21 e.V.: **(N)Onliner Atlas 2009: Eine Topolographi des digitalen Grabens in Deutschland**, Studie der Initiative D21, durchgeführt von TNS Infratest, 2009, http://www.initiatived21.de/category/publikationen/publikationen-2009, (zuletzt besucht am 16.05.2010)

[Van05] J. Vanderdonckt: **A MDA-Compliant Environment for Developing User Interfaces of Information Systems**, Proceedings. Lecture Notes in Computer Science;Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, 2005

[Van08] J. Vanderdonckt: **Model-driven engineering of user interfaces: Promises, successes, failures, and challenges**, Hrsg. S. Buraga, I. Juvina, Proceedings of ROCHI'08, Matrix ROM, Bucarest, 2008, S. 1-10

[VP08] F. Valverde, O. Pastor: **Applying Interaction Patterns: Towards a Model-Driven Approach for Rich Internet Application Developent**, UWWOST 0208: 7th Int. Workshop on Web-Oriented Software Technologies, 2008, S. 7-12

[VS94] G. Verfaillie, T. Schiex: **Solution reuse in dynamic constraint satisfaction problems**, Proceedings of the twelfth national conference on Artifical intelligence, American Assosciation for Artifical Intelligence, Seattle, USA, 1994, S. 307-312

[W3C04] XML Schema: **XML Schema Part 0: Primer Second Edition**, W3C Recommendation 28 October 2004, 2005, http://www.w3.org/TR/xmlschema-0/
(zuletzt besucht am 16.05.2010)

[W3C06] XML: **Extensible Markup Language (XML) 1.0 (Fourth Edition)**, W3C Recommendation 16 August 2006, edited in place 29 September 2006, 2006,
http://www.w3.org/TR/2006/REC-xml-20060816/
(zuletzt besucht am 16.05.2010)

[W3C07] WSDL: **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**, W3C Recommendation 26 June 2007, 2007,
http://www.w3.org/TR/wsdl20/
(zuletzt besucht am 16.05.2010)

[WAP] Open Mobile Alliance: **Wireless Application Protocol Specification 2.0,**
http://www.openmobilealliance.org/tech/affiliates/LicenseAgreement.asp?DocName=/wap/technical_wap2_0-20021106.zip
(zuletzt besucht am 16.05.2010)

[WBB*90] C. Wiecha, W. Bennett, S. Boies, J. Gould, S. Greene: **ITS: a tool for rapid developing interactive applications**, ACM Transaction on Information Systems, ACM, New York, USA, 1990, S. 204-236

[WD08] J. Wright, J. Dietrich: **Survey of existing languages to model interactive web applications**, Conference in Research and Practice in Informatoin Technology Series, Australian Computer Society, Darlinghust, Australia, 2008, S. 113-123

[Weß06] M. Weßendorf: **Struts - Websites mit Struts 1.2 & 1.3 und Ajax effizient entwickeln**, W3L Verlag, Herdecke, 2006

[WL08] H. Wimmel, L. Priese: **Petri-Netze**, Springer Verlag, Berlin, 2008

[WURF] WURFL: **Wireless Universal Resource File (WURFL)**, http://wurfl.sourceforge.net/
(zuletzt besucht am 16.05.2010)

[XUL10] Mozilla Foundation: **XUL**, https://developer.mozilla.org/en/XUL
(zuletzt besucht am 16.05.2010)

[YCJ05] J. S. Yi, Y. S. Choi, J. A. Jacko: **Context Awareness via a Single Device-Attached Accel-erometer During Mobile Computing**, Proceedings of Mobile HCI 2005, 2005

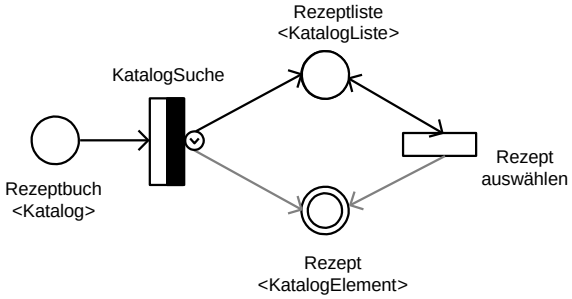
[Zim04] T. Zimmer: **Towards a better understanding of context attributes**, Proceedings of the Second IEEE Annual Conference on Volume, Pervasive Computing and Communications Workshops 2004, 2004, S. 23-27

[ZM90] B. V. Zanden, B. A. Myers, **Automatic, look-and-feel independent dialog creation for graphical user interfaces**, CHI 90: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, USA, ACM, 1990, S. 27-34

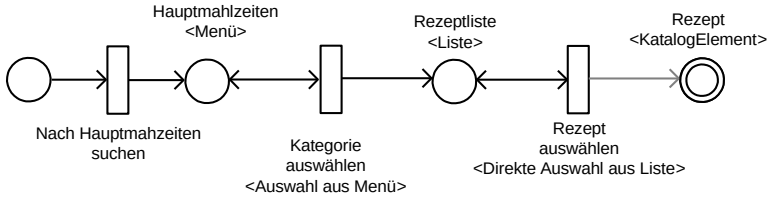
[ZZ99] S. Zhang, .Zhang: **IMC: a method for interval calculus in matrix**, Knowledge and Information Systems, 1999, Vol.1, No. 2, S. 257-268

Anhang A: Objektmetaphern-Katalog

Abstrakt: {[RezeptBuch]}<->RezeptSuchen->{[Rezept]}
Alternative: KategorieSuche
Variante: SuchenImKatalog
AIN: SucheImKatalog_AIN

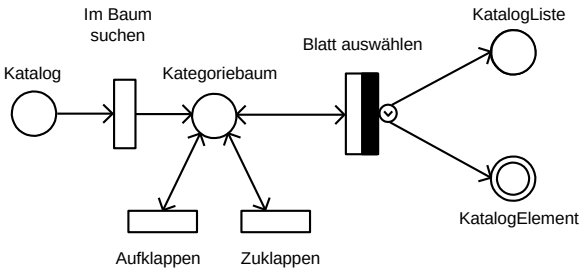


BSF:{KatalogSuche.bsf}
Variante: EinfacheSucheInKategorien
AIN: EinfacheSucheInKategorien_AIN

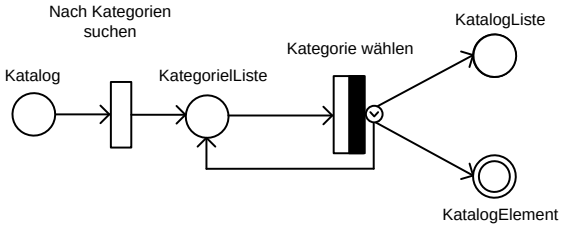


BSF: {}

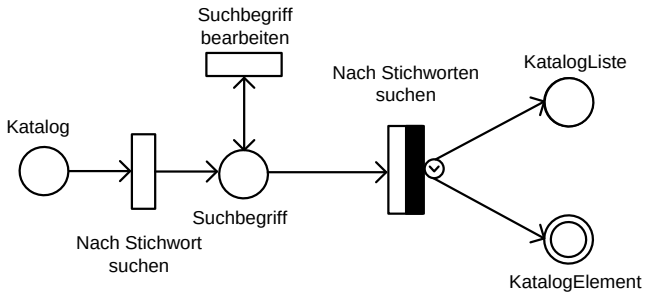
Abstrakt: {[Katalog]}<->KatalogSuche->{[KatalogListe]v[KatalogElement]}
Alternative: SucheInKategorien
Variante: SucheImKategoriebaum
AIN: SucheImKategoriebaum _AIN



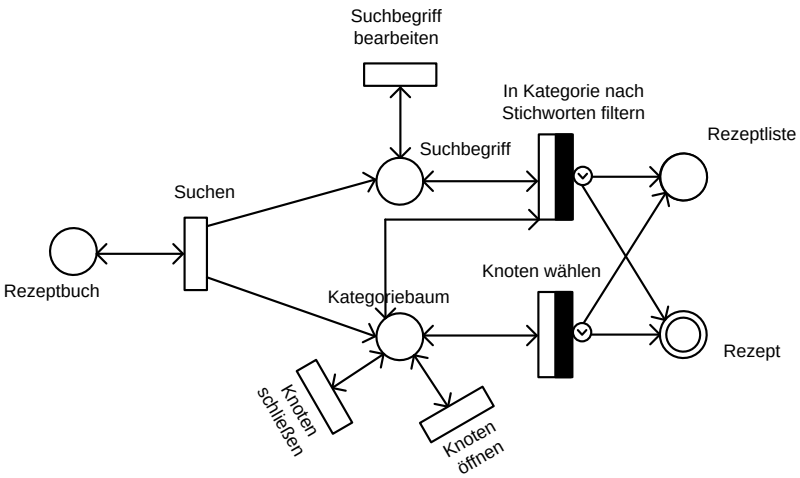
BSF: {SimpleTree.bsf, HierarchischeListe.bsf,
TreeViewCIPhone.bsf}
Variante: ZweistufigeSuche
AIN: ZweistufigeSuche_AIN



BSF: {KatListe.bsf}
Alternative: StichwortSuche
Variante: 1FeldFormular
AIN: 1FeldForm_AIN



BSF: {SuchForm.bsf}
Alternative: Kat&StichwortSuche
Variante: ParalleleKat&Stichwortsuche
AIN: ParalleleKat&1FeldForm_AIN



BSF: {}

Abstrakt: {[Liste]}<->Auswahl 1-N->{[Inhalt]}
Alternative: EinfacheAuswahl
Variante: Auswahl aus Menü
AIN:{}
BSF:{ PullDown_e.bsf
PullDown_k.bsf
Karteireiter.bsf
ButtonMenü.bsf}
Variante: Direkte Auswahl aus Liste

AIN:{}
BSF:{ ButtonListe_sb.bsf
ButtonListe_nav.bsf
LinkListe_nav.bsf
BildListe.bsf
YesNo-question.bsf}
Variante: Indirekte Auswahl aus Liste
AIN:{}
BSF:{ Radiobox.bsf
Checkbox.bsf
ButtonBox.bsf}

Abstrakt: {[Eingabe]}<->Suchbegriff bearbeiten->{[Eingabe]}
Alternative: Freitext eingeben
Variante: Textfeld modifizieren
AIN:{}
BSF:{ InputField.bsf
InputField_mKeyboard}
Variante: Textfeld mit Spracheingabe
AIN:{}
BSF:{ SpeachToText_OK.bsf
SpeachToText_Feedback.bsf}

Abstrakt: {[HierarchischeListe]}<->aufklappen->{[HierarchischeListe]}
Alternative: Blatt aufklappen
Variante: Unterelemente hinzufügen
AIN:{}
BSF:{ Radiobox.bsf
Checkbox.bsf}

Abstrakt: {[HierarchischeListe]}<->zuklappen->{[HierarchischeListe]}
Alternative: Blatt zuklappen
Variante: Unterelemente entfernen
AIN:{}
BSF:{ Radiobox.bsf
Checkbox.bsf}

Abstrakt: {[HierarchischeListe]}<->auswählen->{[HierarchischeListe]v[Inhalt]}
Alternative: Blatt zuklappen
Variante: Unterelemente entfernen
AIN:{}
BSF:{ Radiobox.bsf
Checkbox.bsf}

Abstrakt: {[Liste]}<->Kategorie auswählen->{[Liste]v[Liste]v[Inhalt]}
Alternative: EinfacheAuswahl
Variante: Auswahl aus Menü
AIN:{}
BSF:{ PullDown_e.bsf
PullDown_k.bsf
Karteireiter.bsf
ButtonMenü.bsf}
Variante: Direkte Auswahl aus Liste
AIN:{}

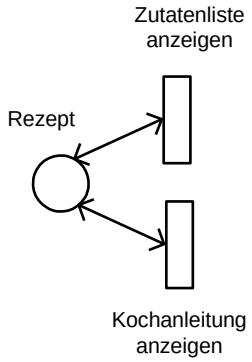
BSF:{ ButtonListe_sb.bsf
ButtonListe_nav.bsf
LinkListe_nav.bsf
BildListe.bsf
YesNo-question.bsf}
Variante: Indirekte Auswahl aus Liste
AIN:{}
BSF:{ Radiobox.bsf
Checkbox.bsf
ButtonBox.bsf}

Abstrakt: {[HierarchischeListe], [Eingabe]}<->In Kategorien nach Stichworten Filtern->{[Liste]v[Inhalt]}
Alternative: Durch Stichwort bestätigen
Variante: Stichwort bestätigen
AIN:{}
BSF:{ InputField&Katbaum_OK.bsf}
Alternative: Durch Kategorieauswahl bestätigen
Variante: Kategorie auswählen
AIN:{}
BSF:{ InputField&Katbaum_KAT.bsf}

Abstrakt: {[Uhr], [Rezept] }->Mahlzeit anlegen->{[Mahlzeit]}
//Trivialinteraktion

Abstrakt: {[Uhr] }->Uhrzeit ändern
Alternative: Durch Stichwort bestätigen
Variante: Stichwort bestätigen
AIN:{}
BSF:{ InputField&Katbaum_OK.bsf}

Abstrakt: {[Rezept] }<->Rezept betrachten
Alternative: Durch das Rezept browsen
Variante: Zwischen Seiten wechseln
AIN:{RezeptBrowsen_AIN}



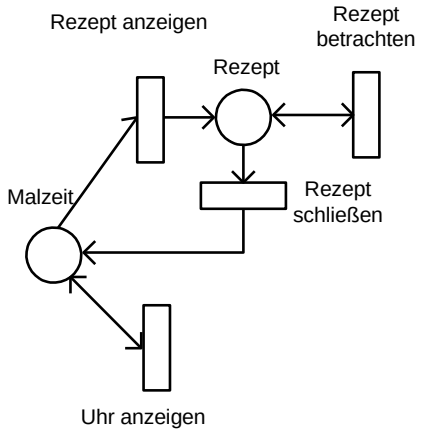
BSF:{}
Alternative: Durch das Rezept scrollen
Variante: Einfaches vertikal skrollen

AIN:{}
BSF:{ Rezept_scrollbar.bsf
Iphone_scrollbar.bsf}

Abstrakt: {[Rezept] }<->Zu den Favoriten hinzufügen

//Trivialinteraktion

Abstrakt: {[Mahlzeit] }<->Mahlzeit betrachten
Alternative: Durch die Mahlzeit browsen
Variante: Rezept&Uhr getrennt anzeigen
AIN:{MahlzeitBrowsen_AIN}



BSF:{Mahlzeit_Übersicht.bsf}
Alternative: Durch die Mahlzeit scrollen
Variante: Einfaches vertikales skrollen
AIN:{}
BSF:{ Mahlzeit_scrollbar.bsf}

Abstrakt: {[Mahlzeit }->Mahlzeit zum Plan hinzufügen->{[Ernährungsplan]}

//Trivialinteraktion

Anhang B: Nutzungsprofile

B.1 „Youngster unterwegs mit dem PDA“

Input-Widget					
→	GUI-Widgets				
	→	Button			
		→	Preference	7	
		→	Image-Label	9	
		→	Text-Label	7	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
	→	Tree			
		→	Preference	9	
		→	H-Tree-Level	9	
		→	Browsable	9	
		→	Text-Label	8	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
		→	Image-Label	9	
	→	Text-Link			
		→	Preference	8	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
	→	Menu			
		→	Preference	9	
			Drop-Down-Only	8	
			Drop-Down-H	9	
			Menu-Bar	9	
		→	Text-Label	7	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
		→	Image-Label	9	
	→	Tabs			
		→	Preference	8	
		→	Text-Label	7	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7

		→	Image-Label	9	
	→	Check-List			
		→	Preference	8	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
	→	Radio-Button			
		→	Preference	7	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
	→	Throttle			
		→	Preference	9	
	→	Link-List			
		→	Preference	8	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
	→	Text-Field			
		→	Preference	7	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
	→	Text-Box			
		→	Preference	8	
			→	Text-Size-Small	9
			→	Text-Size-Medium	8
			→	Text-Size-Large	7
→	Vocal-Widges				
	→	Voice-Comand			
		→	Preference	8	
	→	Speech-to-Text			
		→	Preference	7	
	→	Voice-Recording			
		→	Preference	8	

Tabelle 28: Input-Widget Komponente für „Youngster unterwegs mit dem PDA“

Output-Widgets				
→	GUI-Widgets			
	→	Label		
		→	Preference	8
		→	Text-Size-Small	9
		→	Text-Size-Medium	8
		→	Text-Size-Large	7
		→	Colorset-B/W	5
		→	Colorset-Easy	5
		→	Colorset-Normal	9
	→	Text-Field		
		→	Preference	7
		→	Text-Size-Small	9
		→	Text-Size-Medium	8
		→	Text-Size-Large	7
		→	Colorset-B/W	5
		→	Colorset-Easy	5
		→	Colorset-Normal	9
	→	List		
		→	Preference	8
		→	Text-Size-Small	9
		→	Text-Size-Medium	8
		→	Text-Size-Large	7
		→	Colorset-B/W	5
		→	Colorset-Easy	5
		→	Colorset-Normal	9
	→	Table		
		→	Preference	8
		→	Text-Size-Small	9
		→	Text-Size-Medium	8
		→	Text-Size-Large	7
		→	Colorset-B/W	5
		→	Colorset-Easy	5
		→	Colorset-Normal	9
		→	Grid	9
	→	Image		
		→	Preference	8
	→	Icon		
		→	Preference	9
	→	Process-Bar		
		→	Preference	9
			...	

→	Audio-Widget		
	→	Speach	8
	→	Text-to-Speach	7
	→	Notification-Sounds	9
	→	Background-Music	9
→	Multimedia		
	→	Video	9
	→	Flash	9

Tabelle 29: Output-Widget Komponente für „Youngster unterwegs mit dem PDA“

„Hardware platform“ *Komponente*			
Input			
	Keyboard		
		Preference	9
		Letters	9
		Numbers	9
		Number-pad	9
		Arrows	6
	Mouse		
		Preference	9
		OnScreen-Keyboard	6
		Drag&Drop	9
		Left-Click	9
		Right-Click	9
		Scroller	9
		Special	9
	Touch-Pad (Mouse)	...	
		Preference	8
		OnScreen-Keyboard	6
		Drag&Drop	9
		Left-Click	9
		Right-Click	9
		Scroller	9
		Special	9
	Touch-Display		
		Preference	9
		OnScreen-Keyboard	7
		Drag&Drop	9
		Right-Click	9
		Multi-Touch	9
	Phone-Keypad		

		Preference	7
		Letters	5
		Numbers	9
		Arrows	8
	PDA-ControlKey		
		Preference	7
	Remote-Control	...	
		Preference	7
		Arrows	7
		Letters	6
		Numbers	8
	Voice-Control		
		Preference	9

Tabelle 30: Hardware Komponente für „Youngster unterwegs mit dem PDA“

Layoutpattern		
Base-Area		
	Top-Left	0/0
	Down-Right	320/480
	Scrollbar	Y
	Dynamic	N
	Mandatory	Y
	Sub	B1
	Sub	B2
B1		
	Top-Left	0/0
	Down-Right	320/96
	Scrollbar	N
	Dynamic	N
	Mandatory	Y
B2		
	Top-Left	0/97
	Down-Right	320/480
	Scrollbar	Y
	Dynamic	N
	Mandatory	Y
	Sub	B2.1
	Sub	B2.2
B2.1		
	Top-Left	0/97
	Down-Right	320/384
	Scrollbar	Y
	Dynamic	N
	Mandatory	Y

		Sub	B2.1.1	
		Sub	B2.1.2	
	B2.1.1			
		Top-Left	0/97	
		Down-Right	320/240	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	B2.1.2			
		Top-Left	0/240	
		Down-Right	320/384	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	B2.1			
		Top-Left	0/240	
		Down-Right	320/480	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	Dynamic-Windows	Preference	9	
	Voice			
		Hierarchical	Preference	7
		Command	Preference	9

Tabelle 31: Layoutpattern für „Youngster unterwegs mit dem PDA“

Layoutpattern Komponente				
	Interaction flow			
		sequential	Preference	7
		parallel	Preference	9
		efficient	Preference	7
		mutable	Preference	9
	Layout			
		simple	Preference	7
		complex	Preference	9
	Dynamic windows	Preference	9	
	Scrollbars	Preference	9	

Tabelle 32: Layoutpattern Komponente für „Youngster unterwegs mit dem PDA“

B.2 „Mid Ager bei der Arbeit am PC“

Input-Widget					
→	GUI-Widgets				
	→	Button			
		→	Preference	9	
		→	Image-Label	7	
		→	Text-Label	8	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
	→	Tree			
		→	Preference	8	
		→	H-Tree-Level	6	
		→	Browsable	9	
		→	Text-Label	9	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
		→	Image-Label	9	
	→	Text-Link			
		→	Preference	8	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
	→	Menu			
		→	Preference	9	
			Drop-Down-Only	8	
			Drop-Down-H	7	
			Menu-Bar	7	
		→	Text-Label	9	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
		→	Image-Label	6	
	→	Tabs			
		→	Preference	8	
		→	Text-Label	8	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
		→	Image-Label	7	

	→	Check-List			
		→	Preference	9	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
	→	Radio-Button			
		→	Preference	8	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
	→	Throttle			
		→	Preference	7	
	→	Link-List			
		→	Preference	9	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
	→	Text-Field			
		→	Preference	8	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
	→	Text-Box			
		→	Preference	8	
			→	Text-Size-Small	7
			→	Text-Size-Medium	8
			→	Text-Size-Large	6
→	Vocal-Widges				
	→	Voice-Comand			
		→	Preference	6	
	→	Speech-to-Text			
		→	Preference	6	
	→	Voice-Recording			
		→	Preference	5	

Tabelle 33: Input-Widget Komponente für „Mid Ager bei der Arbeit am PC“

Output-Widgets				
→	GUI-Widgets			
	→	Label		
		→	Preference	8
		→	Text-Size-Small	6
		→	Text-Size-Medium	8
		→	Text-Size-Large	6
		→	Colorset-B/W	5
		→	Colorset-Easy	8
		→	Colorset-Normal	7
	→	Text-Field		
		→	Preference	9
		→	Text-Size-Small	6
		→	Text-Size-Medium	8
		→	Text-Size-Large	6
		→	Colorset-B/W	5
		→	Colorset-Easy	8
		→	Colorset-Normal	7
	→	List		
		→	Preference	8
		→	Text-Size-Small	6
		→	Text-Size-Medium	8
		→	Text-Size-Large	6
		→	Colorset-B/W	5
		→	Colorset-Easy	8
		→	Colorset-Normal	7
	→	Table		
		→	Preference	9
		→	Text-Size-Small	6
		→	Text-Size-Medium	8
		→	Text-Size-Large	6
		→	Colorset-B/W	5
		→	Colorset-Easy	8
		→	Colorset-Normal	7
		→	Grid	9
	→	Image		
		→	Preference	7
	→	Icon		
		→	Preference	8
	→	Process-Bar		
		→	Preference	6
			...	

→	Audio-Widget		
	→	Speach	6
	→	Text-to-Speach	5
	→	Notification-Sounds	5
	→	Background-Music	5
→	Multimedia		
	→	Video	5
	→	Flash	5

Tabelle 34: Output-Widget Komponente für „Mid Ager bei der Arbeit am PC“

„Hardware platform“ *Komponente*			
Input			
	Keyboard		
		Preference	9
		Letters	9
		Numbers	9
		Number-pad	7
		Arrows	6
	Mouse		
		Preference	9
		OnScreen-Keyboard	6
		Drag&Drop	9
		Left-Click	9
		Right-Click	9
		Scroller	9
		Special	5
	Touch-Pad (Mouse)	...	
		Preference	7
		OnScreen-Keyboard	5
		Drag&Drop	5
		Left-Click	9
		Right-Click	9
		Scroller	5
		Special	5
	Touch-Display		
		Preference	6
		OnScreen-Keyboard	5
		Drag&Drop	7
		Right-Click	9
		Multi-Touch	7
	Phone-Keypad		

		Preference	7
		Letters	5
		Numbers	9
		Arrows	8
	PDA-ControlKey		
		Preference	6
	Remote-Control	...	
		Preference	6
		Arrows	8
		Letters	5
		Numbers	8
	Voice-Control		
		Preference	5

Tabelle 35: Hardware Komponente für „Mid Ager bei der Arbeit am PC“

Layoutpattern		
Base-Area		
	Top-Left	0/0
	Down-Right	1280/1024
	Scrollbar	Y
	Dynamic	N
	Mandatory	Y
	Sub	B1
	Sub	B2
	Sub	B3
B1		
	Top-Left	0/0
	Down-Right	1280/205
	Scrollbar	N
	Dynamic	N
	Mandatory	Y
B2		
	Top-Left	0/205
	Down-Right	320/1024
	Scrollbar	Y
	Dynamic	N
	Mandatory	Y
B3		
	Top-Left	320/205
	Down-Right	1280/1024
	Scrollbar	Y
	Dynamic	N
	Mandatory	Y
	Sub	B3.1

		Sub	B3.2	
	B3.1			
		Top-Left	320/205	
		Down-Right	1280/380	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	B3.2			
		Top-Left	320/380	
		Down-Right	1280/1024	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
		Sub	B3.2.1	
		Sub	B3..2.2	
	B3.2.1			
		Top-Left	320/380	
		Down-Right	600/1024	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	B3.2.2			
		Top-Left	600/380	
		Down-Right	1280/1024	
		Scrollbar	Y	
		Dynamic	Y	
		Mandatory	Y	
		Sub	B3.2.2.1	
		Sub	B3.2.2.1	
	B3.2.2.1			
		Top-Left	600/380	
		Down-Right	1280/500	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	B3.2.2.2			
		Top-Left	600/500	
		Down-Right	1280/1024	
		Scrollbar	Y	
		Dynamic	N	
		Mandatory	Y	
	Dynamic-Windows	Preference	9	
	Voice			
		Hierarchical	Preference	7
		Command	Preference	8

Tabelle 36: Layoutpattern für „Mid Ager bei der Arbeit am PC“

Layoutpattern Komponente				
	Interaction flow			
		sequential	Preference	6
		parallel	Preference	7
		efficient	Preference	9
		mutable	Preference	6
	Layout			
		simple	Preference	8
		complex	Preference	7
	Dynamic windows	Preference	8	
	Scrollbars	Preference	7	

Tabelle 37: Layoutpattern Komponente für „Mid Ager bei der Arbeit am PC“

B.3 „Best Ager zuhause am Fernseher“

Input-Widget					
→	GUI-Widgets				
	→	Button			
		→	Preference	8	
		→	Image-Label	6	
		→	Text-Label	8	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
	→	Tree			
		→	Preference	6	
		→	H-Tree-Level	5	
		→	Browsable	6	
		→	Text-Label	8	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
		→	Image-Label	6	
	→	Text-Link			
		→	Preference	8	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
	→	Menu			
		→	Preference	6	
			Drop-Down-Only	8	
			Drop-Down-H	5	

			Menu-Bar	7	
		→	Text-Label	8	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
		→	Image-Label	6	
	→	Tabs			
		→	Preference	8	
		→	Text-Label	8	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
		→	Image-Label	6	
	→	Check-List			
		→	Preference	6	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
	→	Radio-Button			
		→	Preference	7	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
	→	Throttle			
		→	Preference	9	
	→	Link-List			
		→	Preference	8	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
	→	Text-Field			
		→	Preference	7	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8
	→	Text-Box			
		→	Preference	6	
			→	Text-Size-Small	6
			→	Text-Size-Medium	7
			→	Text-Size-Large	8

→	Vocal-Widges			
	→	Voice-Comand		
		→	Preference	8
	→	Speech-to-Text		
		→	Preference	8
	→	Voice-Recording		
		→	Preference	8

Tabelle 38: Input-Widget Komponente für „Best Ager zuhause am Fernseher“

Output-Widgets				
→	GUI-Widgets			
	→	Label		
		→	Preference	8
		→	Text-Size-Small	6
		→	Text-Size-Medium	7
		→	Text-Size-Large	8
		→	Colorset-B/W	5
		→	Colorset-Easy	6
		→	Colorset-Normal	5
	→	Text-Field		
		→	Preference	6
		→	Text-Size-Small	6
		→	Text-Size-Medium	7
		→	Text-Size-Large	8
		→	Colorset-B/W	5
		→	Colorset-Easy	6
		→	Colorset-Normal	5
	→	List		
		→	Preference	6
		→	Text-Size-Small	6
		→	Text-Size-Medium	7
		→	Text-Size-Large	8
		→	Colorset-B/W	5
		→	Colorset-Easy	6
		→	Colorset-Normal	5
	→	Table		
		→	Preference	5
		→	Text-Size-Small	6
		→	Text-Size-Medium	7
		→	Text-Size-Large	8
		→	Colorset-B/W	5

		→	Colorset-Easy	6
		→	Colorset-Normal	5
		→	Grid	6
	→	Image		
		→	Preference	8
	→	Icon		
		→	Preference	6
	→	Process-Bar		
		→	Preference	8
			...	
→	Audio-Widget			
	→	Speach	9	
	→	Text-to-Speach	9	
	→	Notification-Sounds	9	
	→	Background-Music	7	
→	Multimedia			
	→	Video	9	
	→	Flash	9	

Tabelle 39: Output-Widget Komponente für „Best Ager zuhause am Fernseher“

„Hardware platform“				
Komponente				
Input				
	Keyboard			
			Preference	9
			Letters	9
			Numbers	9
			Number-pad	7
			Arrows	6
	Mouse			
			Preference	8
			OnScreen-Keyboard	5
			Drag&Drop	7
			Left-Click	9
			Right-Click	8
			Scroller	6
			Special	5
	Touch-Pad (Mouse)		...	
			Preference	6
			OnScreen-Keyboard	4
			Drag&Drop	4

		Left-Click	9
		Right-Click	6
		Scroller	4
		Special	3
	Touch-Display		
		Preference	8
		OnScreen-Keyboard	6
		Drag&Drop	5
		Right-Click	6
		Multi-Touch	6
	Phone-Keypad		
		Preference	6
		Letters	2
		Numbers	9
		Arrows	6
	PDA-ControlKey		
		Preference	2
	Remote-Control	...	
		Preference	7
		Arrows	8
		Letters	4
		Numbers	7
	Voice-Control		
		Preference	9

Tabelle 40: Hardware Komponente für „Best Ager zuhause am Fernseher“

Layoutpattern		
Base-Area		
	Top-Left	0/0
	Down-Right	800/600
	Scrollbar	N
	Dynamic	N
	Mandatory	Y
	Sub	B1
	Sub	B2
B1		
	Top-Left	0/0
	Down-Right	200/600
	Scrollbar	N
	Dynamic	N
	Mandatory	Y
B2		

		Top-Left	200/0	
		Down-Right	800/600	
		Scrollbar	N	
		Dynamic	N	
		Mandatory	Y	
		Sub	B2.1	
		Sub	B2.2	
	B2.1			
		Top-Left	200/0	
		Down-Right	800/450	
		Scrollbar	N	
		Dynamic	N	
		Mandatory	Y	
	B2.2			
		Top-Left	200/450	
		Down-Right	800/600	
		Scrollbar	N	
		Dynamic	N	
		Mandatory	Y	
	Dynamic-Windows	Preference	4	
	Voice			
		Hierarchical	Preference	9
		Command	Preference	6

Tabelle 41: Layoutpattern für „Best Ager zuhause am Fernseher“

Layoutpattern Komponente				
	Interaction flow			
		sequential	Preference	9
		parallel	Preference	6
		efficient	Preference	6
		mutable	Preference	5
	Layout			
		simple	Preference	9
		complex	Preference	7
	Dynamic windows	Preference	5	
	Scrollbars	Preference	5	

Tabelle 42: Layoutpattern Komponente für „Best Ager zuhause am Fernseher“

Anhang C: Funktionen zum Aufbau des BSF-AIN

Datenstrukturen:

{}:	leere Menge
BSF (BSF):	ein Benutzerschnittstellenfragment
AIN (AIN):	ein AIN
ABSTRAKT Abstrakte Interaktion:	eine Abstrakte Interaktion (im OM-Katalog)
ALTERNATIVE Alternative:	eine Alternative (im OM-Katalog)
VARIANTE Variante:	eine Variante (Im OM-Katalog)
IM (AINTERAKTION v ALTERNATIVE v VARIANTE):	Interaktionsmetapher (entweder eine Abstrakte Interaktion, Alternative oder Variante)
IM[] {IM_1,...,IM_n}:	eine Menge von Interaktionsmetaphern
BSF-AIN[] {(IM_1,BSF_1),..,(IM_n,BSF_n)}:	Menge von Paaren aus IM und BSF
BSF/AIN (BSF v AIN):	Datenstruktur, die entweder ein BAF oder ein AIN ist
PROFIL Nutzungsprofil:	Datenstruktur die ein Nutzungsprofil beschreibt
OM-KATALOG OM-Katalog:	Datenstruktur die den OM-Katalog beschreibt

---

// Aufgabe dieser Funktion ist Erstellung einer Abbildung von einem AIN auf ein BSF-AIN[],
also eine Datenstruktur, welche die Abbildung beschreibt

(BSF-AIN, AIN) bildeBSF-AIN (OM-KATALOG omKatalog, PROFIL nutzungsprofil, AIN ain)

IM[] interaktionen = ain.AlleInteraktionen() *//liefert alle Interaktionsmetaphern des AINs*

BAF-AIN[] abbildung = {}

BSF/AIN ergebnis = {}

WHILE (interaktionen.count() != 0) DO //Schleife solange durchlaufen bis die Liste leer ist.

ergebnis = analysiereInteraktion(omKatalog, nutzungsprofil, interaktionen[1])

IF (ergebnis.istBSF()) *//Wenn ergebnis vom Typ BSF ist liefert die Bedingung ein „wahr“*

THEN abbildung = abbildung + (interaktionen[1], ergebnis)

ELSE *//Das „ergebnis“ ist ein AIN*
ain = erweitereAIN(ain, interaktionen[n], ergebnis)
//Ersetzte die Interaktion „interaktion[n]“ durch das AIN im „ergebnis“

interaktionen = interaktionen.addInteraktionen(ergebnis.AlleInteraktionen())
*//Liste um alle Interaktionsmetaphern des AIN ergebnis erwe*itern

interaktionen = interaktionen.entferne(interaktionen[1])
//Entfernt die übergebene Interaktion aus der Liste

RETURN (abbildung, ain)

// Aufgabe dieser Funktion ist Analyse einer Interaktion (sowohl Abstrakte Interaktion, Alternative oder Variante) Das Ergebnis ist entweder ein BSF das die Interaktion realisiert, oder ein AIN, das diese genauer spezifiziert.

(BSF/AIN) analysiereInteraktion
(OM-KATALOG omKatalog, PROFIL nutzungsprofil, IM interaktion)

IF (omKatalog.istAbstrakteInteraktion(interaktion)) *//Interaktion ist eine „Abstrakte Interaktion“*

THEN RETURN (analysiereAbstrakteInteraktion(omKatalog, nutzungsprofil, interaktion))

IF (omKatalog.istAlternative(interaktion)) *//Interaktion ist eine „Alternative“*

THEN RETURN (analysiereAlternative(omKatalog, nutzungsprofil, interaktion))

IF (omKatalog.istVariante(interaktion)) *//Interaktion ist eine „Variante“*

THEN RETRUN (analysiereAbstrakteInteraktion(omKatalog, nutzungsprofil, interaktion))

// Aufgabe dieser Funktion ist Analyse einer Abstrakten Interaktion. Das Ergebnis ist entweder ein BSF das die Interaktion realisiert, oder ein AIN, das diese genauer spezifiziert.

(BSF/AIN) analysiereAbstrakteInteraktion
(OM-KATALOG omKatalog, PROFIL nutzungsprofil, IM interaktion)

BSF/AIN ergebnis = {}

IM[] alternativen = omKatalog.alleAlternativen(interaktion)
// Liefert alle Alternativen zur übergebenen Interaktion

alternativen = sortiereAlternativen(alternativen)
//Sortiere die Liste der Alternativen anhand der durch die Funktion: bewerteBSF/AIN(analysiereAlternative()) definierten Gewichtung

// Definieren einer Default-Lösung:
Ausgewählt wird die besten Alternative und auch die Zweitplazierte, falls diese einen Schwellwert erreicht.

ergebnis = analysiereAlternative(omKatalog, nutzungsprofil, alternativen[1])
//Bestes AIN oder BSF wird immer ausgewählt

IF (bewerteBSF/AIN(analysiereAlternative(omKatalog, nutzungsprofil, alternativen[2]))>5)
// “wahr“, falls die zweitbeste Alternative den Schwellenwert überschreitet

THEN ergebnis = komponiereAlternativen (alternativen[1]), alternativen[2])
//komponiereAlternativen komponiert zwei Interaktionen zu einem AIN

// Ende der Default-Lösung

// Nur eine Alternative wird angeboten:
Ein strikt sequentieller Interaktionsablauf ist gefordert
IF (Nutzungsprofil. Interactionflow. sequential.Preference < 7
& Nutzungsprofil. Interactionflow. parallel.Preference > 3
& Nutzungsprofil. Interactionflow. mutable.Preference <3)

THEN ergebnis = analysiereAlternative(omKatalog, nutzungsprofil, alternativen[1])
// Ende der sequentiellen Lösung

// Wird ein flexibler Interaktionsablauf gefordert, sollen möglichst viele
// Alternativen angeboten werden. Auch hier wird ein Schwellwert angesetzt.

```
IF      (Nutzungsprofil. Interactionflow. sequential.Preference < 5
        & Nutzungsprofil. Interactionflow. parallel.Preference > 7
        & Nutzungsprofil. Interactionflow. efficient.Preference >5
        & Nutzungsprofil. Interactionflow. mutable.Preference  > 7)

        THEN

                ergebnis = alternativen[1]

                int n=1

                // Durchlaufe alle Alternativen und füge alle hinzu, die den Schwellwert überschreiten
                WHILE ( n < alternativen.count()) DO

                        IF (bewerteBSF/AIN(anaysiereAlternative(omKatalog, nutzungsprofil,
                        ..... alternativen[n+1])>5)

                                THEN ergebnis = komponiereAlternativen
                                                        (ergebnis, alternativen[n+1])

                        n = n+1

// Ende der flexiblen Lösung

RETURN ergebnis
```

// Aufgabe dieser Funktion ist Analyse einer Alternative. Das Ergebnis ist entweder ein BSF das die Alternative realisiert, oder ein AIN, das diese genauer spezifiziert.

```
(BSF/AIN) analysiereAlternative
                        (OM-KATALOG omKatalog, PROFIL nutzungsprofil, IM interaktion)

BSF/AIN ergebnis = {}

IM[] varianten = omKatalog.alleVarianten(interaktion)
                                // Liefert alle Varianten zur übergebenen Interaktion

// Setze die erste Variante in der Liste als beste
Ergebnis = anaysiereVariante(omKatalog, nutzungsprofil, varianten[1])

int n=1

// Durchlaufe alle Varianten und finde bessere.
WHILE  n < varianten.count()) DO

        IF (bewerteBSF/AIN(omKatalog, nutzungsprofil,
                        anaysiereVariante(omKatalog, nutzungsprofil, varianten[n])) <
        bewerteBSF/AIN(omKatalog, nutzungsprofil,
                        anaysiereVariante(omKatalog, nutzungsprofil, varianten[n+1])))

                THEN ergebnis = anaysiereVariante(omKatalog, nutzungsprofil, varianten[n+1])

        n = n+1

RETURN ergebnis
```

// Aufgabe dieser Funktion ist Analyse einer Variante. Das Ergebnis ist entweder ein BSF das die Variante realisiert, oder ein AIN, das diese genauer spezifiziert.
(BSF/AIN) analysiereVariante
(OM-KATALOG omKatalog, PROFIL nutzungsprofil, IM interaktion)

```
BSF/AIN ergebnis = {}

    IF (omKatalog.hatBSF(interaktion) ) // "wahr" falls zur Variante ein BSF im OM-Katalog vorliegt

        THEN

            BSF[] variantenBSF = omKatalog.alleBSF(interaktion)
                                    // Liefert alle BSF zur übergebenen Interaktion

            // Setze das erste BSF in der Liste als beste Lösung
            ergebnis = variantenBSF[1]

            int n=1

            // Durchlaufe alle BSF und finde bessere Lösungen.
            WHILE ( n < variantenBSF.count()) DO

                IF (bewerteBSF/AIN(omKatalog, nutzungsprofil, variantenBSF[n]) <
                      (bewerteBSF/AIN(omKatalog, nutzungsprofil, variantenBSF[n+1])

                    THEN ergebnis = varianteBSF[n+1]

                n = n+1


// Aktuell liegt das best geeignete BSF im Ergebnis. Nun muss überprüft werden, ob ggf. ein AIN
        eine bessere Lösung darstellt.
IF (omKatalog.hatAIN(interaktion)) // „wahr" falls zur Variante ein AIN im OM-Katalog verfügbar ist.

    THEN

        IF (omKatalog, nutzungsprofil, bewerteBSF/AIN(ergebnis) <
                                          bewerteBSF/AIN(omKatalog, nutzungsprofil, ain))
                                        // „wahr" falls das AIN eine bessere Lösung darstellt.

            THEN Ergebnis = {ain}

RETURN ergebnis
```

// Aufgabe dieser Funktion ist es zu ermitteln, ob ein übergebenes BSF oder AIN bezogen auf das Nutzungsprofil geeignet ist. Das Ergebnis ist eine rationale Zahl im Bereich 0..9, wobei 0 als unpassend und 9 als ideal interpretiert wird.
(Eignung) bewerteBSF/AIN
(OM-KATALOG omKatalog, PROFIL nutzungsprofil, BSF/AIN bsf/ain)

```
BSF-AIN abbildung ) {}

int ergebnis = 0

// Wenn ein BSF  übergeben wurde
IF (bsf/ain.istBSF()) // „wahr" falls bsf/ain ein BSF ist

    THEN

        Ergebnis = bewerteBSF(Nutzungsprofil, bsf/ain)
```

```
// Wenn ein AIN übergeben wurde
IF (bsf/ain.istBSF()) // „wahr“ falls bsf/ain ein AIN ist

        THEN

                abbildung = bildeBSF-AIN
                        (OM-KATALOG omKatalog, PROFIL nutzungsprofil, AIN ain)

                int n=1

                // Durchlaufe alle Elemente und summiere die Bewertungen aller BSF.
                WHILE ( n <=  abbildung.count()) DO

                        ergebnis = ergebnis +
                                        bewerteBSF/AIN(omKatalog, nutzungsprofil, abbildung[n].BSF)

                        n = n+1

                        ergebnis = ergebnis div abbildung.count()
                                        //Berechne den Durchschnitt der Wertungen aller BSF

RETURN ergebnis
```

// Aufgabe dieser Funktion ist die Interaktion „interaktion“ im AIN „ausgangsAIN“ durch das AIN „interaktionsAIN“ zu ersetzten.
(AIN) erweitereAIN (AIN ausgangsAIN, IM interaktion, AIN interaktionsAIN)

// Diese Funktion realisiert die in Abschnitt 5.3.2 beschriebenen komplexen Transitionen, wobei das „interaktion“ die komplexe Transition im AIN ausgangsAIN ist. Beschrieben wird die komplexe Transition durch das AIN „interaktionsAIN“.

// Aufgabe dieser Funktion ist es zwei Interaktionen, die Alternativen einer Abstrakten Interaktion sind, zu einem AIN zusammen zu führen.
(BSF/AIN) komponiereAlternativen(IM alternative1, IM alternative2)

// Die Komposition zweier Alternativen kann als trivial angesehen werden, da die beiden Alternativen über die gleichen Eingangs- und Ausgangsstellen verfügen. Abbildung 41 veranschaulicht die Komposition an einem Beispiel mit zwei Eingangs- und zwei Ausgangsstellen.

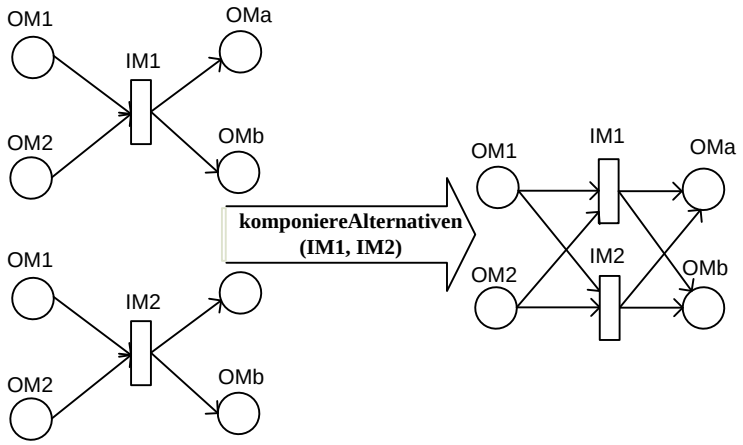


Abbildung 41: Funktion: (BSF/AIN) komponiereAlternativen (IM, IM)

// Aufgabe dieser Funktion ist es zu ermitteln, ob ein übergebenes BSF bezogen auf das Nutzungsprofil geeignet ist. Das Ergebnis ist eine rationale Zahl im Bereich 0..9, wobei 0 als unpassend und 9 als ideal interpretiert wird.
(Eignung) bewerteBSF(PROFIL nutzungsprofil, BSF bsf)

// Gibt an, ob das BSF auf das durch (nutzungsprofil.UAProf) beschriebene Endgerät überhaupt dargestellt werden kann
{0,1} geräteModifikator = 0

// Beschreibt die Eignung der Input-Widgets des BSF bezogen auf das Nutzungsprofil
real inputWidgets = 0

// Beschreibt die Eignung der Output-Widgets des BSF bezogen auf das Nutzungsprofil
real outputWidgets = 0

// Beschreibt die Eignung der Hardware- des BSF bezogen auf das Nutzungsprofil
real HardwarePlattform=0

// Beschreibt die Eignung nutzungsprofil.LayoutComponent in Bezug auf bsf.Nutzungskontextvektor.LayoutComponent. Je näher die Werte zusammen liegen, desto besser ist die Eignung.
real layoutComponent = 0

//Berechne des GeräteModifikators
geräteModifikator = * *Wende auf die Spezifikationen (XAML) aller Präsentationsfragmente des Benutzerschnittstellenfragmentes den Parser für das verwendete Endgerät (beschrieben durch Nutzungsprofil.UAProf) an. Wenn für alle XAML ein gültiges Ergebnis generiert wird setze „1“, ansonsten „0“*

// Berechnung der Eignung der Input-Widgets
inputWidgets =* *Im Nutzungskontextvektor des BSF sind alle Input-Widgets, die in einem der Präsentationsfragmenten des BSF vorkommen, mit einer 1 gekennzeichnet (alle anderen mit einer 0). Zur Berechnung der Eignung werden alle im BSF vorkommenden*

Input-Widgets mit der Präferenz aus dem Nutzungsprofil Multipliziert und der Durchschnitt der Werte gebildet. Der Durchschnitt definiert „inputWidget“.

// Berechnung der Eignung der Output-Widgets
outputWidgets =* *Im Nutzungskontextvektor des BSF sind alle Output-Widgets, die in einem der Präsentationsfragmenten des BSF vorkommen, mit einer 1 gekennzeichnet (alle anderen mit einer 0). Zur Berechnung der Eignung werden alle im BSF vorkommenden Output-Widgets mit der Präferenz aus dem Nutzungsprofil Multipliziert und der Durchschnitt der Werte gebildet. Der Durchschnitt definiert „outputWidget“.*

Berechnung der Eignung der Eingabe- und Ausgabemedien
hardwarePlattform =* *Input- und Output-Widgets sind für jedes Endgerät mit Eingabe- und Ausgabemedien der Hardwareplattform verknüpft. Aus dem im Nutzungskontextvektor des BSF vorkommenden Input- und Output-Widgets, lässt sich damit ableiten, welche Eingabe- und Ausgabemedien benutzt werden. Diese werden mit einer 1 gekennzeichnet. Die Berechnung erfolgt analog zur inputWidget und outputWidget.*

// Berechnung der Eignung des Layout-Aufbaus
LayoutComponent = * *Berechnet wird für jeden Parameter bsf.Nutzungskontextvektor.LayoutComponent und nutzungsprofil.LayoutComponent: 9 minus die Differenz der beiden Parameter. Der Durchschnitt dieser Werte definiert die Eignung der LayoutComponent*

// Die Eignung wird anhand der Folgenden Formel berechnet. Dabei werden alle Parameter gleich gewichtet.

$$ergebnis = \left(\frac{inputWidget}{4} + \frac{outputWidgett}{4} + \frac{hardwarePlattform}{4} + \frac{layoutComponent}{4}\right) * geräteModifikator$$

RETURN ergebnis

Anhang D: Grundlagen für die Komposition von Benutzerschnittstellenfragmenten

Dieser Abschnitt wurde als Exkurs angelegt, der sich intensiv mit der Beschreibung von Benutzerschnittstellenfragmenten auseinandersetzt und ein besseres Verständnis der Komposition von Benutzerschnittstellenfragmenten ermöglicht.

In einem ersten Schritt erfolgt eine Betrachtung von Start- und Endpunkten von Benutzerschnittstellenfragmenten. Dieser Betrachtung folgend wird gezeigt werden, wie die Durchführung eines Benutzerschnittstellenfragmentes als Sequenz aus Benutzerinteraktionen spezifiziert werden kann. Die Analyse dieser Sequenzen haben für die Kompositionsmöglichkeiten von Benutzerschnittstellen erhebliche Relevanz hat. In einem zweiten Schritt wird gezeigt, wie sich solche Sequenzen automatisiert aus einem BSF ermitteln lassen. Diese Analysen ermöglichen es, Daten zu ermitteln, die aus den Benutzerinteraktionen (bspw. Texteingabe, Auswahl von Elementen, usw.) extrahiert werden können. Diese Daten beschreiben einen Input für nachfolgende Benutzerschnittstellenfragmente und sind daher für die Komposition von Benutzerschnittstellenfragmenten von Bedeutung.

D.1 Start und Ende eines Benutzerschnittstellenfragmentes

Ein Benutzerschnittstellenfragment kann als Intervall aufgefasst werden, das durch Benutzerinteraktion beschrieben wird. Dabei kann angenommen werden, dass Benutzerinteraktionen sequentiell erfolgen.

Nur multimodale Benutzerschnittstellen (vgl. [NC93]) ermöglichen es mehrere Interaktionen gleichzeitig durchzuführen, indem unterschiedliche Modalitäten (Gestik und Sprache, oder Mausbewegung und Tastatureingabe) gleichzeitig oder überlappend erfolgen. Solche Interaktionen lassen sich aber dennoch in eine Sequenz überführen, wenn eine multimodale Interaktion als Einheit abgebildet wird, auch wenn sie aus unterschiedlichen Modalitäten bestehen sollte.

Ein Benutzerschnittstellenfragment lässt sich aber nicht auf ein Intervall reduzieren, vielmehr beinhaltet es viele unterschiedliche Intervalle, die abhängig vom Nutzerverhalten sind. Einerseits ist nicht vorhersehbar, in welcher Reihenfolge ein Benutzer bestimmte Interaktionen durchführt, andererseits können Benutzerschnittstellenfragmente auch unterschiedliche Pfade und alternative Abläufe beschreiben. Die erste Herausforderung ist somit den Beginn und das Ende eines Benutzerschnittstellenfragmentes zu identifizieren. In einem zweiten Schritt (in Anhang D.2) werden dann die möglichen Intervalle innerhalb eines BSF untersucht, um Aussagen über Inhalte und Daten innerhalb des BSF treffen zu können.

Auch wenn nicht vorhersehbar ist, aus welchen Nutzerinteraktionen (Intervallen) das BSF bei der Durchführung des Benutzerschnittstellenfragmentes bestehen wird, lassen sich zumindest Start- und Endpunkte des BSF bestimmen:

Die Startpunkte des BSF sind durch die Startdialogfragmente eines Benutzerschnittstellenfragmentes gegeben. Diese sind laut Spezifikation eindeutig definiert und somit immer eindeutig bestimmbar.

Ein Benutzerschnittstellenfragment gilt bei der Durchführung als aktiv, wenn mindestens ein Präsentationsfragment existiert, das den Status „aktiviert“ trägt. Solange ein Benutzerschnittstellenfragment aktiv ist, können Benutzerinteraktionen erfolgen und das BSF ist nicht abgeschlossen.

Ein BSF gilt in seiner Durchführung als abgeschlossen, sobald alle Präsentationsfragmente den Status „deaktiviert“ erreicht haben. In diesem Zustand können keine weiteren Benutzerinteraktionen erfolgen, da keines der Präsentationsfragmente mehr verfügbar und damit für den Benutzer zugänglich ist. Entsprechend definiert die Benutzerinteraktion, die den Status des letzten Präsentationsfragmentes auf „deaktiviert“ setzt, das Ende des BSF. Da jede Benutzerinteraktionen über ein Event einem Dialogfragment zugeordnet werden kann, beschreibt mindestens ein Dialogfragment das Ende der Ausführung des Benutzerschnittstellenfragmentes.

Um Benutzerschnittstellenfragmente als Intervalle beschreiben zu können, muss jedes BSF auch ein oder mehrere definierte Enden besitzen. Dabei können mehrere Dialogfragmente in der Lage sein, das Benutzerschnittstellenfragment zu beenden. Die Menge aller Dialogfragmente, die das Benutzerschnittstellenfragment beenden können, definiert die Enden des Interaktionsprozesses und somit des Benutzerschnittstellenfragmentes. Es kann nicht eindeutig vorhergesagt werden, wann und welches Dialogfragment das Benutzerschnittstellenfragment beendet, wohl aber welche Möglichkeiten es gibt, die zu einem Ende führen können. Jede dieser Möglichkeiten muss bei der Komposition von Benutzerschnittstellen Beachtung finden.

Das Ende der Ausführung eines Benutzerschnittstellenfragmentes wird durch ein oder mehrere Dialogfragmente beschrieben. Die Ausführung jedes dieser Dialogfragmente führt dazu, dass kein Präsentationsfragment mehr den Status „aktiviert“ trägt und beschreibt damit das Ende der Ausführung des Benutzerschnittstellenfragmentes. Diese Dialogfragmente werden im Folgenden Enddialogfragmente genannt und die Events (und damit Benutzerinteraktionen), die diese auslösen Schlussinteraktionen.

Eine automatisierte Bestimmung von Enddialogfragmenten ist nicht trivial, da unterschiedliche Nutzerinteraktionen zu einem Beenden führen können. Zu beachten ist weiterhin, dass unterschiedlichen Enddialogfragmenten eines Benutzerschnittstellenfragments auch eine semantische Bedeutung zugewiesen wird. Ein Benutzerschnittstellenfragment beschreibt nicht notwendigerweise nur Interaktionsabläufe, die zu einem eindeutigen Ergebnis führen. So kann dasselbe Benutzerschnittstellenfragment Interaktionen für unterschiedliche Aufgaben bereitstellen, oder aber alternative Ausgänge einer Aufgabe modellieren.

Aus der Perspektive des BSF können unterschiedliche Dialogfragmente existieren, welche die Definition eines Endfragmentes erfüllen, aber das Benutzerschnittstellenfragment mit unterschiedlicher Semantik

beenden. Betrachtet man beispielsweise das BSF „Nach Stichwort suchen“ aus dem BSF-AIN in Abbildung 46 („Mahlzeit erstellen“ für den Nutzungskontext „Youngster unterwegs mit dem PDA“), so definiert das BSF zwei mögliche Ausgänge („Rezept“ oder „Rezeptliste“). Nachfolgende Benutzerschnittstellenfragmente müssen in der Lage sein zu unterscheiden, welches Ergebnis nun eingetreten ist. Um beide Fälle innerhalb eines BSF unterscheidbar zu machen, muss das Benutzerschnittstellenfragment auch zwei Enddialogfragmente enthalten, welche Aufschluss über die beiden möglichen Ausgänge des Interaktionsprozesses geben.

Im BSF-AIN wird ein Benutzerschnittstellenfragment durch seine Eingangsstellen, die Transition und die Ausgangsstellen repräsentiert. Eingangsstellen des BSF-AINs beschreiben dabei Präsentationsfragmente und sind Bestandteile des BSF. Ein Startdialogfragment aktiviert das Benutzerschnittstellenfragment, was übertragen auf die Darstellung des BSF im BSF-AIN durch die Markierung einer Eingangsstelle abgebildet wird. Laut der gegebenen Definition gilt das BSF folglich als aktiviert. Besitzt eine Transition mehr als eine Eingangsstelle, so muss für jede Eingangsstelle ein Dialogfragment existieren, das diese aktiviert. Weiterhin wird dadurch auch eine Bedingung formuliert, wann die Durchführung des Interaktionsablaufes des Benutzerschnittstellenfragmentes fortgesetzt werden kann. Erst wenn alle Eingangsstellen markiert wurden können die Interaktionen im BSF fortgesetzt werden.

Im BSF-AIN wird durch die Ausgangsstellen einer Transition das Beenden eines BSF abgebildet. Obwohl die Ausgangsstellen selbst nicht Teil des BSF sind, spezifizieren jedoch die Kanten zu den Ausgangsstellen Enddialogfragmente. AINs verwenden weiterhin für die Beschreibung der Ausgangsstellen einer Transition die „AND“ and „OR“ Verknüpfung (vgl. „komplexe Stellen“ in Abschnitt 5.3.2). Jeder „AND“-Verknüpfung (und damit allen Kanten der Verknüpfung) im AIN wird genau ein Enddialogfragment zugeordnet, wobei jede Kante einer „OR“-Verknüpfung ein eigenes Enddialogfragment zugeordnet wird. Dass bedeutet also, dass jede Variante, die zu einer anderen Markierung nach dem Schalten einer Transition führt, durch ein eigenes Enddialogfragment abgebildet wird. Systeminteraktionen, bei denen nicht vorhergesagt werden kann, wie das Ergebnis der Schaltvorganges aussieht, werden ebenfalls durch „AND“- und „OR“-Verknüfungen markiert und somit ebenfalls durch unterschiedliche Enddialogfragmente beschrieben.

D.2 Interaktionspfade in Benutzerschnittstellenfragmenten

Die Beschreibung von Benutzerschnittstellenfragmenten als Intervall bedarf weiterhin der Einführung des Begriffes des Interaktionspfades zur Beschreibung unterschiedlicher Interaktionswege im Benutzerschnittstellenfragment:

> *Ein Interaktionspfad eines Benutzerschnittstellenfragments ist definiert als eine Abfolge von Benutzerinteraktionen, die zu einem Beenden der Ausführung des Benutzerschnittstellenfragmentes führen. Spezifiziert ist ein Interaktionspfad als eine Sequenz aus Dialogfragmenten, wobei ein Interaktionspfad immer mit dem Startdialogfragment startet und mit einem der Enddialogfragmente endet.*

Auch ein Benutzerschnittstellenfragment mit nur einem Enddialogfragment kann laut Definition mehrere Interaktionspfade beschreiben, da unterschiedliche Interaktionsfolgen aber auch unterschiedliche Wege im Interaktionsprozess zum selben Enddialogfragment führen können. Interaktionspfade mit demselben Enddialogfragment können in Bezug auf die zu ermittelnden Interaktionsdaten als gleichwertig aufgefasst werden, in dem Sinne, dass jeder dieser Interaktionspfade aus Sicht des Nutzers zum gleichen Ergebnis führt. Daraus lässt sich folgern, dass die Menge an Informationen, die auf Interaktionspfaden mit gleichem Enddialogfragment ermittelt werden kann, eine gemeinsame Teilmenge besitzt. Diese Teilmenge definiert, welche Informationen an ein nachfolgendes Benutzerschnittstellenfragment übergeben werden können, ohne dass jeder Interaktionspfad für die Komposition gesondert betrachtet werden muss. Für die Komposition von Benutzerschnittstellenfragmenten ist eine Analyse von Interaktionspfaden unerlässlich, da mit deren Hilfe Aussagen über das Verhalten und die Schnittstellen von Benutzerschnittstellenfragmenten getroffen werden können

In einem ersten Schritt wird ein Algorithmus präsentiert, der eine automatische Berechnung aller zyklenfreien Interaktionspfade eines Benutzerschnittstellenfragmentes auf der Basis der Spezifikation eines BSF ermöglicht. Die Menge aller Interaktionspfade bildet die Grundlage für eine Analyse der Interaktionsabläufe, die durch ein Benutzerschnittstellenfragment beschrieben werden.

Die Ausgangslange für die Untersuchung ist die Spezifikation eines Benutzerschnittstellenfragments, wie es in Abschnitt 7.2.2.1 definiert wurde. Abbildung 101 visualisiert eine vereinfachte graphische Darstellung eines Benutzerschnittstellenfragmentes, das sich auf die Beschreibung von Dialog- und Präsentationsfragmenten beschränkt und nur die Aktivierung von Präsentationsfragmenten im Interaktionsablauf beschreibt.

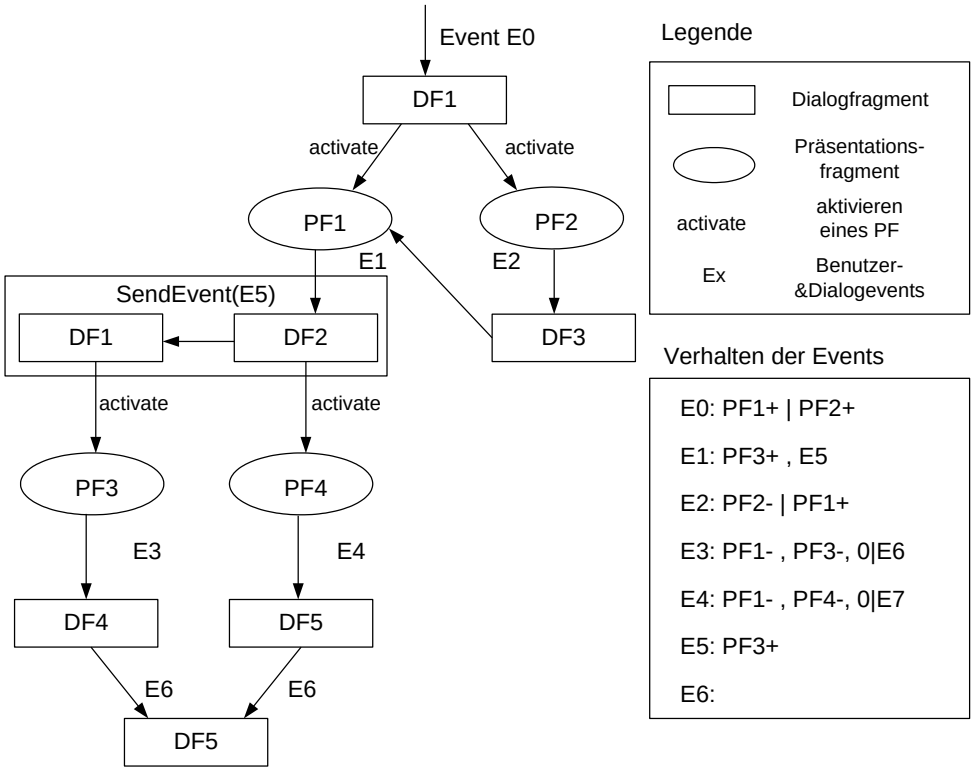


Abbildung 101: Interaktionsablauf in der Dialogspezifikation

Dargestellt wird einerseits, welche Auswirkungen Dialogfragmente auf die Sichtbarkeit von Präsentationsfragmenten haben, und andererseits, wie sich die Sichtbarkeit von Präsentationsfragmenten auf mögliche Benutzerinteraktionen und damit auf die Möglichkeiten zur Ausführung von Dialogfragmenten auswirken. Rechtecke repräsentieren Dialogfragmente und Ellipsen Präsentationsfragmente. Die Kanten sind gerichtet, wobei eine Kante von einem Dialog- zu einem Präsentationsfragment das Aktivieren (also einen Statuswechsel im PF) beschreibt, während eine Kante in entgegengesetzter Richtung das Auslösen eines Events durch eine Benutzerinteraktion oder durch die SendEvent-Direktive in einem Dialogfragment repräsentiert. Letztere Kantenvariante wird mit dem auslösenden Event annotiert. Alle im Graphen dargestellten Beziehungen können aus der Spezifikation des Benutzerschnittstellenfragments automatisch abgeleitet werden.

Jeder Interaktionsablauf beginnt mit der Ausführung des Startdialogfragmentes (DF1 in Abbildung 101). Dieses kann in dem gegebenen Beispiel die Präsentationselemente PF1 und PF2 aktivieren. Beide stellen wiederum Möglichkeiten zur Benutzerinteraktion zur Verfügung, die durch Events modelliert werden. Jedes Event wird wiederum von einem DF bearbeitet, das weitere Präsentationsfragmente aktivieren kann. Das genaue Verhalten von DF, die aufgrund von Events ausgelöst wurden, ist in Abbildung 101 rechts unten beschrieben. Im Folgenden wird die Notation zur Beschreibung des Verhaltens eingeführt, und beschrieben, wie dieses automatisch aus der Spezifikation des BSF ermittelt werden kann.

Die Interaktionsmöglichkeiten des Benutzers werden durch die Verfügbarkeit von Präsentationsfragmenten beschrieben, da diese die möglichen Interaktionen mit der Benutzerschnittstelle definieren. Daher erfolgt die Analyse möglicher Interaktionspfade auf Basis der Sichtbarkeit von Präsentationsfragmenten und den Interaktionsmöglichkeiten, die mit diesen verbunden sind.

In einem ersten Schritt wird das Verhalten von Dialogfragmenten untersucht. Hierzu werden alle Direktiven eines Dialogfragments betrachtet. Das Ziel der Untersuchung ist alle Auswirkungen auf die Sichtbarkeit von Präsentationsfragmenten zu erfassen. Dabei werden alle Direktiven vernachlässigt, die keine Auswirkungen auf die Sichtbarkeit von Präsentationsfragmenten haben. Diesen Anforderungen entsprechend werden ausschließlich die Direktiven „aktivieren“, „deaktivieren“, „sendEvent“ und das IF-Konstrukt betrachtet. Auch wenn das IF-Konstrukt keine direkte Auswirkungen auf die Sichtbarkeit von PF hat, muss es dennoch betrachtet werden, wenn zumindest in einem Zweig eine der Direktiven („aktivieren“, “deaktivieren“, “sendEvent“) enthalten ist, da dann das Konstrukt sehr wohl Auswirkungen auf die Sichtbarkeit haben kann. Folgende Notation wird für die Beschreibung der Auswirkungen eines DF auf die Sichtbarkeit von Präsentationsfragmenten eingeführt:

- PF+
  Beschreibt das Aktivieren eines Präsentationsfragments PF.
- PF-
  Beschreibt das Deaktivieren eines Präsentationsfragments PF.
- ϕ
  Beschreibt eine leere Direktive. Diese wird beispielsweise für die Beschreibung eines IF-Konstrukts benötigt, um anzuzeigen, dass in einem Zweig keine Auswirkungen auf die Sichtbarkeit von PF zu erwarten sind.
- E
  Beschreibt die SendEvent-Direktive und damit den Aufruf eines Dialogfragmentes, das für die Verarbeitung des Events E verantwortlich ist. Ein Event ist demnach ein Platzhalter, der für Direktiven eines anderen Dialogfragments eingesetzt wird.
- <Direktive1>,<Direktive2>
  Beschreibt, dass sowohl Direktive1 als auch Direktive2 ausgeführt werden. Da Direktiven durch eine Sequenz beschrieben werden, ist dies die Basisverknüpfung von Direktiven. „,“ ist dabei assoziativ und kommutativ. Es gilt dabei PF+,PF+ = PF+ und PF+, PF-= ϕ .
- <Direktive1>|<Direktive2>
  Beschreibt das IF Konstrukt. Dabei wird entweder Direktive1 oder Direktive2 ausgeführt. Sollte ein Zweig keine Direktive enthalten, die Auswirkungen auf die Sichtbarkeit von PF hat, so wird diese durch die ϕ beschrieben. Geschachtelte IF Konstrukte werden durch Klammern gekennzeichnet.

Bedingt durch das IF Konstrukt kann die Verarbeitung eines Events unterschiedliche Auswirkungen auf die Interaktionsmöglichkeiten haben. So kann das Statement PF1+,PF2+|(PF3+,PF1-) zu folgenden Ergebnissen führen: (PF1+,PF2+) oder (PF3+). Für Letzteres gilt: (PF1+,PF3+,PF1-) = (PF3+). Auf eine genauere Betrachtung der Kondition des IF Konstruktes wird also verzichtet, stattdessen wird angenommen, dass beide Ereignisse eintreten können.

Benutzerinteraktionen werden durch Events modelliert. Welche Interaktionen möglich sind, hängt dabei von den zur gleichen Zeit verfügbaren Präsentationsfragmenten ab. Die Analyse von Interaktionspfaden beinhaltet somit die Betrachtung der Mengen aller sichtbaren Präsentationsfragmente und der Interaktionsmöglichkeiten, die diese dem Benutzer bieten.

Die Berechnung von Interaktionspfaden erfolgt mit Hilfe eines sogenannten BSF-Interaktionsbaums. Die Knoten eines BSF-Interaktionsbaumes werden durch eine Menge von sichtbaren Präsentationsfragmenten repräsentiert, während die Kanten Events beschreiben, die mit der gegebenen Menge von PF möglich sind. Die Wurzel des BSF-Interaktionsbaumes ist die leere Menge. Das Event, welches das Startdialogfragment auslöst, beschreibt die von der Wurzel ausgehenden Kanten. Grundlage für die Generierung eines BSF-Interaktionsbaumes ist die Spezifikation eines BSF, wobei eine Beschreibung der Auswirkungen von Events bereits in der oben eingeführten Notation vorliegt.

Ein BSF-Interaktionsbaum wird nach folgenden Regeln aufgebaut:

> *(1) Die Wurzel des BSF-Interaktionsbaumes ist die leere Menge.*
>
> *(2) Der Wert eines Knotens wird durch eine Menge von PF bestimmt. Laut Spezifikation ist jedem PF eine Menge an Events zugeordnet, die mögliche Benutzerinteraktionen beschreiben. Die Menge aller Events der PF definiert die Kinder des Knotens. Dabei gilt, dass jedes Event mindestens ein Kind definiert. Ein Event kann mehr als ein Kind definieren, wenn in seiner Verarbeitung IF-Konstrukte verwendet werden, die Auswirkungen auf die Sichtbarkeit haben. In diesem Fall wird für jede mögliche Variante ein Kind erzeugt.*
>
> *(3) Der Wert eines Knotens berechnet sich aus dem Wert seines Vaters und dem zugewiesenen Event. Für die Berechnung kann die eingeführte Notation zur Beschreibung der Auswirkungen von DF auf die Sichtbarkeit von PF benutzt werden. Dabei gilt, dass PF+ das Hinzufügen und PF- das Entfernen eines PF aus der Menge der sichtbaren PF beschreibt.*

(4) Die Kinder der Wurzel sind über das Startdialogfragment definiert und können analog zu (2) berechnet werden.Abbildung 102 visualisiert einen BSF-Interaktionsbaum, der basierend auf dem Modell in Abbildung 101 erstellt wurde.

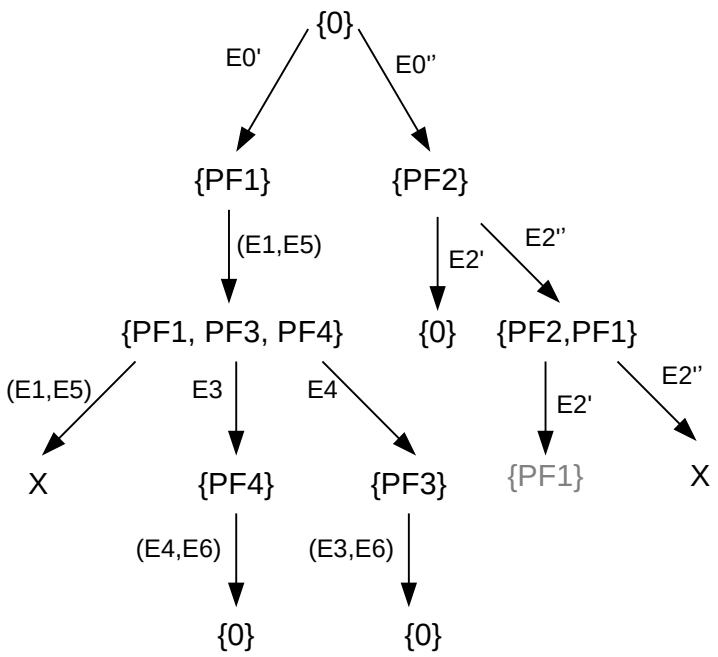


Abbildung 102: BSF-Interaktionsbaum

Die Wurzel des Baumes ist die leere Menge. Das Startdialogfragment, das durch E0 beschrieben ist, definiert die Kinder der Wurzel. E0 (definiert durch (PF1+|PF2+)) beschreibt zwei mögliche Kinder {PF1} (beschrieben durch das Event E0'=PF1+) und {PF2} (beschrieben durch das Event E0''=PF2+). Jedes Event und auch jede Variante eines Events ist somit eindeutig unterscheidbar. Auf jedes Blatt eines Knotens kann nun Regel (2) für die Berechnung der Kinder angewendet werden, und zur Berechnung des Wertes der Kinder die Regel (3).

Um zu gewährleisten, dass ein BSF-Interaktionsbaum zyklenfrei bleibt, muss bei der Berechnung der Kinder eines Knotens folgende Bedingung berücksichtigt werden:

> *Für einen Knoten werden keine Kinder berechnet, wenn der Knoten entweder die leere Menge aber nicht die Wurzel ist, oder wenn der Wert des Knotens (die Menge der Präsentationsfragmente) bereits im BSF-Interaktionsbaum existiert, und dessen Kinder bereits berechnet wurden (grau hinterlegte Knoten in Abbildung 102).*

Die Bedingung zur Vermeidung von Zyklen basiert auf der Annahme, dass Knoten mit gleichem Wert identisch sind, und daher nur einer dieser Knoten weiter analysiert werden muss. Dabei gilt auch, dass Blätter, die durch Wiederholung des gleichen Events generiert wurden, aus dem Baum entfernt werden (vgl. „X“ in Abbildung 102). Dies ist immer dann der Fall, wenn Vater und Sohn durch das gleiche Event generiert wurden.

Interaktionspfade lassen sich durch eine Analyse der Pfade im Baum berechnen. Zur Ermittlung der Interaktionspfade wird im BSF-Interaktionsbaum der Pfad von der Wurzel zu den Blättern mit dem Wert ϕ

analysiert. Wird dabei ein Blatt erreicht, das einen Wert ungleich ϕ besitzt, so wird der Pfad in einem Knoten mit gleichem Wert fortgesetzt. Bei der Pfadanalyse ist zu beachten, dass Pfade, aber auch Zyklen auftreten können, die niemals zu einem Blatt mit ϕ führen. Solche Pfade sind nach Definition keine Interaktionspfade, da sie zu keinem definierten Ende des Benutzerschnittstellenfragmentes führen. Solche Pfade werden nicht weiter betrachtet.

Folgende Interaktionspfade sind durch den in Abbildung 102 dargestellten BSF-Interaktionsbaum beschrieben:

*Pfad1: {E0',(E1,E5),E3,(*E4,*E6)}*

Pfad2: {E0',(E1,E5),E4,(E3,E6)}

Pfad3: {E0´´,E2'}

Pfad4: {E0´´,E2´´,E2´,(E1,E5),E3,(E4,E6)}

Pfad5: {E0´´,E2´´,E2´,(E1,E5),E4,(E3,E6)}

Schlussinteraktionen bestimmten sich dabei aus dem letzen Event, dass ausgeführt wird. In dem beschriebenen Beispiel ist dies E2 und E6. Dabei beschreibt E6 ein Event, das durch eine SendEvent-Direktive ausgelöst wurde und damit nicht direkt einem Präsentationsfragment zugeordnet werden kann. Dieser Mechanismus wird speziell verwendet, um unterschiedlichen Interaktionspfaden dieselbe Schlussinteraktion zuzuordnen.

Der folgende Abschnitt beschreibt einige Besonderheiten der Dialogspezifikation, die zur Veranschaulichung der allgemeinen Methodik ausgeblendet wurden.

D.3 Dynamische Präsentationsfragmente und „nicht aktive" Interaktionsobjekte

Die Analyse von Interaktionspfaden vernachlässigt auf den ersten Blick einige Besonderheiten der Spezifikation von BSF. So ist eine Vorraussetzung für die Abbruchbedingung zur Vermeidung von Zyklen eine endliche Menge von PF. Dynamische PF, die über die „Erzeugen"-Schnittstelle eines PF (vgl. Abschnitt 7.2.2.2) erstellt werden können, stehen zu dieser Annahme im Widerspruch. Über die „Erzeugen"-Schnittstelle wird ein PF mit einem Namen generiert, der erst zur Laufzeit vergeben wird, und daher nicht aus der Dialogspezifikation ermittelt und analysiert werden kann. Aus der Dialogspezifikation ist nur ersichtlich, dass nicht ein Name für die Aktivierung solch eines PF verwendet wird, sondern ein Parameter, der über ein Event übergeben wurde.

Für eine Analyse von Interaktionspfaden kann aber davon ausgegangen werden, dass jede Instanz eines dynamischen PF gleichwertig in Bezug auf die gebotenen Interaktionsmöglichkeiten ist. Folglich definieren unterschiedliche Instanzen keine neuen Interaktionsmöglichkeiten und sind daher zu vernachlässigen.

In Szenarien, in welchen ein Benutzer mehrere Instanzen eines PF anlegt, und den Interaktionsprozess mit einer dieser Instanzen fortsetzt, ohne die anderen zu schließen, können die übrigen Instanzen vernachlässigt werden. Zwar definieren diese immer noch die Interaktionsmöglichkeiten der Instanz, deren Ausführung würde aber keine neuen Interaktionsmöglichkeiten eröffnen, da sie nur den existierenden Interaktionsablauf überlagern würden.

Für die Pfadanalyse können alle dynamische PF auf eine Instanz reduziert werden, und wie oben beschrieben verfahren werden. Bei der Analyse der Statements der DF ist aber zu berücksichtigten, das kein PF-Name zur Adressierung verwendet wird, sondern ein Wert, der über das auslösende Event übergeben wird. Da aber eine Zuordnung von Events zu PF existiert, kann der entsprechende PF-Name eindeutig ermittelt werden.

Ein weiterer Mechanismus, der in der bisherigen Pfadanalyse keine Berücksichtigung gefunden hat, ist die Möglichkeit von DF Interaktionsobjekte in PF zu deaktivieren und wieder zu aktivieren. Diese Änderung hat direkte Auswirkungen auf die Interaktionsmöglichkeiten, da die Deaktivierung eines Interaktionsobjektes die Interaktionsmöglichkeiten des Benutzers mit einem PF modifiziert. Folglich müssen PF mit unterschiedlichen Aktivierungen der Interaktionsobjekte eindeutig unterschieden werden. Weiterhin führt auch die Aktivierung oder Deaktivierung von Interaktionaktionselementen zu Veränderungen der Interaktionsmöglichkeiten und ist somit Bestandteil der Interaktionspfadanalyse. Die in Anhang D.2 eingeführte Pfadanalyse muss um folgende Betrachtung erweitert werden:

Jedes PF wird neben seinem Namen durch die aktivierten Events (Interaktionsobjekte) beschrieben. Nur aktivierten Events werden im BSF-Interaktionsbaum Kinder zugeordnet. Das Aktiveren bzw. Deaktivieren von Interaktionsobjekten wird mit in die Beschreibung der Auswirkungen der Ausführung von Dialogfragmenten einbezogen und wirkt sich folgendermaßen auf die Ermittlung des Wertes eines Knotens im BSF-Interaktionsbaum aus:

Seien PF1(E1, E3) ein Präsentationsfragment mit den aktivierten Events E1 und E3. Das Aktivieren des Events E2 hat zur Folge, dass das Präsentationsfragment PF1(E1, E3) deaktiviert wird und PF1(E1, E2, E3) aktiviert wird.

Durch die Unterscheidung von PF anhand ihrer Events und der Einbeziehung der Aktivierung und Deaktivierung von Interaktionsobjekten in den Analyseprozess, kann die in Anhang D.2 eingeführte Methodik weiterhin eingesetzt werden.

D.4 Erste Ansätze für eine Analyse von Interaktionsdaten der Schnittstellen von Benutzerschnittstellenfragmenten

Interaktionspfade beschreiben alle zyklenfreien Interaktionsabläufe, die zu einem Ende der Ausführung des Benutzerschnittstellenfragments führen. Jeder Interaktionspfad definiert somit eine mögliche Schnittstelle für das Benutzerschnittstellenfragment, das für eine Komposition betrachtet werden muss.

Die Schnittstellen eines Benutzerschnittstellenfragmentes definieren sich folgendermaßen:

> *Ein Benutzerschnittstellenfragment verfügt über mindestens eine Schnittstelle. Jede dieser Schnittstellen wird durch ein Enddialogfragment gekennzeichnet. Alle Interaktionspfade, die zum selben Enddialogfragment führen, werden in einer Schnittstelle zusammengefasst. Jeder Schnittstelle wird weiterhin eine Menge von Datensätzen zugeordnet, die im Zuge der Durchführung der Interaktionen erhoben wurden. Diese Daten stehen potentiell für ein nachfolgendes Benutzerschnittstellenfragment zur Verfügung.*

Dem Beispiel in Abbildung 102 sind somit folgende Schnittstellen zuzuordnen:

- E6
  beschrieben durch Pfade 1, 2, 4 und 5.
- E2
  beschrieben durch Pfad 3.

Einige Regeln zur Reduktion der zu betrachtenden Pfade lassen sich bereits auf der Ebene der Event-Sequenzen definieren. Dabei ist zu beachten, dass nur Interaktionpfade einer Schnittstelle betrachtet werden:

- Existiert nur ein Pfad, so legt dieser die Interaktionsdaten für die Schnittstelle fest.
- Zwei Pfade lassen sich verschmelzen, wenn der eine Pfad eine Permutation des anderen Pfades ist. In beiden Pfaden werden dieselben Interaktionen getätigt, lediglich die Reihenfolge ist eine andere.
- Seien X und Y die Menge aller Events zweier Pfade. Wenn X eine Teilmenge von Y ist, definiert X die kleinste gemeinsame Datenbasis von X und Y.

Durch die Anwendung dieser drei Regeln lassen sich die Pfade 1 und 2 und die Pfade 4 und 5 für die Schlussinteraktion E6 zusammenfassen.

Die Ermittlung einer kleinsten gemeinsamen Datenbasis und schließlich der Daten einer Schnittstelle erfordert die Betrachtung der Daten, die im Zuge der Durchführung der Interaktionen ermittelt werden. Ausgangslage für die Ermittlung der Daten einer Endschnittstelle bildet die Analyse der Operationen auf SessionData, dem DataSet (als gemeinsamer Pool von Datenstrukturen) und die Interaktionspfade der Schnittstelle.

Die Datenstruktur SessionData enthält Datensätze, die von Dialogfragmenten mit dem Ziel des Datenaustausches zwischen anderen Dialogfragmenten abgelegt werden. SessionData enthält somit ausschließlich Datensätze, die vom Modellierer des Dialogfragmentes als relevant und potentiell für andere Dialogfragmente nutzbar identifiziert wurden. Damit können die Datensätze der SessionData als Ergebnis der Durchführung eines Dialogfragmentes aufgefasst werden.

SessionData wird in Form einer Hashtable realisiert, und enthält Datensätze in der Struktur (Name, Datenstruktur), wobei Name ein Bezeichner ist und die Datenstruktur ein Element der Menge DataSet ist. Der Zugriff auf die SessionData ist auf die beiden Direktiven (Name, Datastructure)→Sessiondata.put und Name→Sessiondata.get→Datastructure beschränkt, die von den Dialogfragmenten durchgeführt werden können.

In einem ersten Schritt werden alle Daten erfasst, die bei der Durchführung eines Interaktionspfades auflaufen. Hierzu werden die Direktiven aller Dialogfragmente analysiert, die den Events des Interaktionspfades zugeordnet sind. Zu berücksichtigen ist weiterhin, dass Events nicht nur das Dialogfragment bestimmen, sondern gegebenenfalls auch noch welcher alternative Zweig des BSF durchlaufen wird. Dies ist dann der Fall, wenn IF-Direktiven enthalten sind, die Auswirkungen auf die Sichtbarkeit von Präsentationsfragmenten haben (vgl. Anhang D.2). Durch die Betrachtung der SessionData.put Direktiven kann eine Datenmenge aufgebaut werden, welche bei der Ausführung des Interaktionspfades anfällt. IF-Direktiven müssen auch hier gesondert betrachtet werden. Nur wenn sichergestellt ist, dass in jedem Ausführungsfall dieselbe (definiert durch den Parameter Name) SessionData.put Direktive ausgeführt wird, wird der Datensatz auch betrachtet, da anderenfalls nicht sichergestellt ist, dass diese Daten immer erhoben werden. Beispiele, in denen ein Datensatz immer zur Verfügung steht, dies aber nicht zuverlässig automatisiert erkannt werden kann, sind leicht zu konstruieren. Um komplexe Analysen der einzelnen IF Direktiven zu vermeiden (deren Erfolg nicht garantiert werden kann), sollte eine Richtlinie für die Entwickler von Dialogfragmenten existieren, die vorsieht, dass Daten, die von einem Dialogfragment immer bereitgestellt werden, gesondert deklariert werden. Dies könnte beispielsweise realisiert werden, indem zu Beginn der Ausführung des Dialogfragmentes SessionData.put Direktiven mit leeren Daten für alle Datensätze eingefügt werden.

Für jeden Interaktionspfad kann auf diesem Wege eine Menge von Datensätzen ermittelt werden. Um eine gemeinsame Datenmenge für eine Schnittstelle zu bestimmen, sind die Daten aller Interaktionspfade zu betrachten, die zum selben Enddialogfragment führen. Ermittelt werden soll die maximale Menge an Daten, die allen betrachteten Interaktionspfaden zugrunde liegen. Da nicht sichergestellt ist, dass auf den unterschiedlichen Interaktionspfaden die gleichen Datenstrukturen verwendet wurden, muss eine Analyse in der Lage sein, unterschiedliche Datenstrukturen zu berücksichtigen. Zusätzlich kann ein und dieselbe Datenstruktur auch mehrfach in der Menge vertreten sein, in diesem Fall ist lediglich der Name das Unterscheidungsmerkmal.

In einem gewissen Rahmen (bspw. Ermittlung von Datenstrukturen), lässt sich diese Analyse automatisiert durchführen, wobei die Qualität des Ergebnisses stark von dem verwendeten Datenmodell abhängig ist, das durch DataSet definiert ist. Hier ist zu fordern, dass möglichst viele Datenstrukturen durch Komposition definiert werden. Der Einsatz einer gemeinsamen Menge aus Datenstrukturen und die Komposition von Benutzerschnittstellenfragmenten einer Domäne, ermöglichen erst den Einsatz einfacher und automatisierbarer Mechanismen für die Datenanalyse. Schließlich beschreibt DataSet auch nur einen Ausschnitt eines allgemeinen Datenmodells, der auf die Beschreibung von Interaktionen reduziert ist.

Weiterhin ist anzumerken, dass lediglich Datenstrukturen automatisiert verglichen werden. Informationen, die gegebenenfalls in den Bezeichnern der Datenstrukturen in SessionData enthalten sind, lassen sich nur durch semantische Annotationen ermitteln.

Anhang E: XML-basierte UIDLs

XML-basierte UIDL auf Auszug aufs der UsiXML Web-Page

Auszug aus der Liste von XML-basierten User Interface Description Languages (UIDL) und Systeme, wie sie auf der UsiXML Web-Page unter http://www.usixml.org/index.php?mod=pages&id=58 zu finden ist. Der Auszug wurde am 28.03.2010 erstellt.

- AAIML (Alternate Abstract Interface Markup Language) developed by Trace Center, University of Wisconsin, USA.
- AUIL (Abstract User Interface Language) is developed by the Alcatel S.A.
- AUIML (Abstract User Interface Markup Language) is developed by IBM Corp., USA.
- CCXML (Call Control eXtensible Markup Language) is designed to provide telephony call control support for VoiceXML or other dialog systems.
- D3ML (Device-Independent MultiModal Mark-up Language) is a language developed in the framework of the SNOW Consortium (Services for NOmadic Workers).
- DISL (Dialog and Interface Specification Language) is developed by Cooperative Computing and Communication Laboratory, University of Paderborn, Germany.
- EMMA (Extensible MultiModal Annotation markup language) is developed by the World Wide Web Consortium.
- eNode (element Node) is developed by eNode, Inc., USA.
- GIML (Generalized Interface Markup Language) is used for interface description in GITK (Generalized Interface Toolkit) project and is developed by Stefan Kost as Ph.D. Thesis at Technical University of Dresden and Leipzig University of Applied Sciences, Germany.
- GladeXML allows dynamic loading of user interfaces from XML descriptions. It is part of the LibGlade library.
- IDEAL (Interface Description Language?) developed by Telefonica I+D in the context of the MyMobileWeb and MORFEO projects. Two versions are available: V1.0 and V2.0.
- IDS Use of XML (Interaction Design System Use of XML) developed by Honeywell Labs, USA.
- IMML (Interactive Message Modeling Language) is a language proposed by Jair Leite to the designers describe an user interface as a message
- InkML is designed by the ink subgroup of the Multi Modal Interaction Working Group of W3C
- InTml (Interaction Techniques Markup Language) is developed by Departement of Computing Science, University of Alberta, Canada.
- 3dml (3D Markup Language) is developed by Departement of Computing Science, University of Alberta, Canada.
- ISML (Interface Specification Meta-Language) is developed by Simon Crowle in his Ph.D. Thesis at Bournemouth University, UK.
- Luxor (XML UI Language Toolkit), Luxor Team

- MAWL (The Mother of All Web Languages) is developed by Bell Laboratories, Lucent Technologies.
- MDML (Multiple Device Markup Language) is developed by School of Computer Science, Telecommunications and Information Systems, DePaul University, USA.
- MPML (Multimodal Presentation Markup Language) is an XML-based language developed to enable the description of multimodal presentation using character agents in easier way.
- MRML (Multimedia Retrieval Markup Language) is supported by the Viper Project.
- MXML (Macromedia Flex Markup Language) is developed by Macromedia Flex Developer Center.
- OpenLaszlo is developed by Laszlo Systems, Inc., USA.
- Plastic ML is developed by Trigone Laboratory-CUEEP, Universit? des Sciences et Technologies de Lille, France.
- Prado (PHP Rapid Application Development Object-oriented) is an event-driven and component-based framework for PHP 5.
- SeescoaXML (Software Engineering for Embedded Systems using a Component-Oriented Approach) is developed by EDM-LUC, Belgium.
- RIML (Renderind Independent Markup Language) is supported by Consensus Project.
- Sisl (Several Interfaces, Single Logic) is developed by Bell Laboratories, Lucent Technologies.
- SSIML (Scene Structure and Integration Modelling Language) is developed by Media Informatics Group, University of Munich, Germany.
- SunML (Simple Unified Natural Markup Language) is a language for wirting user interfaces which are device independent.
- UIML (User Interface Markup Language) is developed by the UIML Consortium, involving Harmonia, Inc., USA. and Virginia Tech.
- VHML (Virtual Human Markup Language) is designed to accommodate the various aspects of Human-Computer Interaction with regards to Facial Animation, Body Animation, Dialogue Manager interaction, Text to Speech production, Emotional Representation plus Hyper and Multi Media information.
- VoiceXML (Voice Extensible Markup Language) is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations.
- VRML (Virtual Reality Modeling Language) allows to create "virtual worlds" networked via the Internet and hyperlinked with the World Wide Web.
- XAMJ is developed by XAMJ Working Group as part of SourceForge.net initiative.
- XAML (Microsoft Extensible Application Markup Language) developed by Microsoft, Corp., USA.
- XDL (XML Interface Description Language) developed by Georgia Institute of Technology.
- XICL (extensible markup language for developing user interface and components) is developed by Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Brazil.

- XIML (eXtensible Interface Markup Language) is developed by the XIML Forum, lead by RedWhale Soft. Corp., USA.
- XIN-XML language to build and integrate distributed applications.
- XISL(eXtensible Interaction Scenario Language) is developed by Toyohashi University of Technology, Nitta Laboratory, Japan.
- XMMVR (eXtensible markup language for MultiModal interaction with Virtual Reality worlds) is realated to concepts like: virtual reality, multimodal interaction (graphical and vocal), XML technolgies and Java development.
- XUL (XML-based User Interface Language) is developed by the Mozilla Fundation

Glossar

Abstract Interaction Net	Abstract Interaction Nets (kurz AIN) stellen eine formale Beschreibungsmethodik bereit, die es dem Modellierer erlaubt, Interaktionen (mit der Benutzerschnittstelle) mit Hilfe von **Objekt-** und **Interaktionsmetaphern** zu modellieren. AINs ermöglichen eine Spezifikation von Benutzerschnittstellen, ohne Aussagen über das verwendete Endgerät, den Nutzer und die Umgebung der Nutzung treffen zu müssen. Sie dienen zur Spezifikation der **Interaktionsbeschreibung**. AINs basieren auf **Dialognetzen.**
Abstrakte Interaktionen	Abstrakte Interaktionen fassen im **Objektmetaphern-Katalog Interaktionsmetaphern** zusammen, die aus Sicht des Benutzers zur gleichen Zielverfolgung benutzt werden. Abstrakte Interaktionen ermöglichen eine Klassifizierung von **Interaktionsmetaphern**. Siehe **Alternativen** und **Varianten**.
Adaptive Benutzerschnittstellen	Der in dieser Arbeit verwendete Begriff der Adaptivität von Benutzerschnittstellen nimmt die Zielsetzung von Thevenin und Coutaz (vgl. **„plastic user interfaces“** in [TC99]) als Ausgangslage, nämlich dass adaptive Benutzerschnittstellen in der Lage sein sollten, sich effizient (möglichst automatisiert) an äußere Rahmenbedingungen anzupassen.
Adipositas	Adipositas beschreibt ein starkes Übergewicht (ab BMI>30).
Adipositas-Begleiter	Der Adipositas-Begleiter dient als Anwendungsbeispiel, an welchem die Generierung von Benutzerschnittstellen exemplarisch durchgeführt wird. Der Adipositas-Begleiter beschreibt ein informationslogistisches telemedizinisches System, das in der Nachsorgephase (in häuslicher Umgebung) einer stationären **Adipositas**-Therapie eingesetzt wird.
Alternativen (Interaktionen)	Alternativen fassen im **Objektmetaphern-Katalog Interaktionsmetaphern** zusammen, die auf unterschiedlichen Methoden beruhen, aber zum selben Ergebnis führen. Einer **Abstrakten Interaktion** sind 1...n Alternativen zugeordnet. Siehe **Abstrakte Interaktionen** und **Varianten**.
Anwendungsschnittstelle	Die Anwendungsschnittstelle spezifiziert die Schnittstellen eines **Benutzerschnittstellenfragmentes** zu externen Datenquellen (bspw. Web-Services), auf die **Präsentationsfragmente** und **Dialgfragmente** zugreifen können

Ausgabemedien	Physische Objekte zur Realisierung von Ausgaben wie beispielsweise Display, Audio, Sprachausgabe, usw. (vgl. **Interaktionsmedien**).
Benutzerinteraktionen	Siehe **Interaktionen**.
Benutzerschnittstellen-fragment	Benutzerschnittstellenfragmente spezifizieren die wieder verwendbare Teile von Benutzerschnittstellen und ihre Möglichkeit zur Komposition. Sie kapseln das Wissen über die Realisierung der Benutzerschnittstelle und beinhalten auch Wissen darüber, wie diese sich verhalten und welche Schnittstellen sie anbieten. Ein Benutzerschnittstellenfagmente wird spezifziert durch seine **Präsentations-fragmente**, **Dialogfragmente** und die **Anwendungsschnittstelle**. Benutzerschnittstellenfragmente realisieren die **Modellfragmente** des „**Compositional Modeling“**.
Bewertungsalgorithmus	Im Rahmen dieser Arbeit wird so ein Algorithmus genannt, der es ermöglicht, die Eignung von unterschiedlichen **Benutzerschnittstellenfragmenten** für die Realisierung einer Interaktion bezogen auf ein gegebenes **Nutzungsprofil** zu bewerten.
BSF-AIN	Beschreibt ein Modell, das im Rahmen des Generierungsprozesses eingesetzt wird. Das BSF-AIN nutzt die Spezifikation eines **Abstract Interaction Nets**, für das eine Abbildung auf die **Benutzerschnittstellenfragemente** der „Domain Theory“ angegeben werden kann.
Compliance	Mit Complicance wird die Therapietreue eines Patienten beschrieben.
Compositional Modeling	Das „Compositional Modeling“ wurde als Leitmotiv für die Generierung von Benutzerschnittstellen gewählt. Ursprünglich wurde dieser Begriff von Falkenhainer und Forbus in [FF91] eingeführt. Im Kern beschreibt das „Compositional Modeling“ eine wissensbasierte Interferenzmaschine, die basierend auf einer **Szenario-beschreibung** und einer domänenspezifischen Wissensbasis (**Domain Theory**) in der Lage ist, ein Szenario (das **Scenario Model**) aufzubauen, das auf eine gegebene Fragestellung (in [FF91] **Query** genannt) hin optimiert ist.
Default-Profil	Default-Profile für die Komponenten Gerät, Nutzer und Umgebung der Nutzung sind Bestandteile eines **Nutzungskontextmodells.** Default-Profile fassen eine Menge von typischen Anforderungen zusammen, und definieren somit eine Informationsbasis, die weiter verfeinert werden kann.

Desktop-PC	Stationärer Heim-Computer besthend aus einem Gehäuse, Bildschirm, Maus und Tastatur.
Dialogfragment	Dialogfragmente spezifizieren die Dialogsteuerung eines **Benutzerschnittstellenfragmentes**. Jedes Dialogfragment beschreibt die Reaktion des **Benutzerschnittstellenfragmentes** auf Ereignisse, die von **Präsentationsfragmenten**, der **Anwendungsschnittstelle** aber auch von anderen Dialogfragmenten ausgelöst werden können. Es existiert eine eindeutige Zuordnung von Ereignissen zu Dialogfragmenten. Jedes Dialogfragment reagiert auf genau ein Ereignis und besitzt eine eigene Verarbeitungslogik, welche die Aktivitäten des Dialogfragmentes beschreibt, die beim Eintreten eines Ereignisses durchgeführt werden. Die Beschreibung der Verarbeitungslogik liegt in einer Pseudo-Programmiersprache vor, die in unterschiedlichen Implementierungen interpretiert werden kann.
Dialogmodelle	Beschreiben Interaktionen eines Benutzers mit einer Benutzerschnittstelle mit Hilfe von **Dialogspezifikationen**.
Dialognetze	Dialognetzen (vgl. [BFJ96]) sind eine auf Petri-Netzen basierende Modellierungssprache für Fenster-basierte graphische Benutzerschnittstellen. Dialognetze liefern die formale Grundlage für die Definition von **Abstract Interaction Nets**.
Dialogspezifikationen	Beschreiben unterschiedliche Modellierungsmethodiken wie Zustandsübergangsdiagramme, Petrinetze, Graphtransformationen, Grammatiken, aber auch Event-basierte Beschreibungssprachen und Constraints. Ziel ist die Spezifikation einer Benutzerschnittstelle.
Domain Theory	Definiert im Rahmen des „**Compositional Modeling“** eine eine Menge von **Modellfragmenten** (in [FF91] Domänenfragmente genannt) und Regeln über ihren Einsatz. Im Rahmen der vorliegenden Arbeit wird der Begriff „Domain Theory“ beibehalten, und beschreibt eine Menge von **Benutzerschnittstellenfragmenten**, die für die Generierung der Benutzerschnittstelle als Wissensbasis herangezogen wird.
Domänenspezifischer Baukasten	Der „Domänenspezifische Baukasten“ wurde im Rahmen des Entwicklungsprozesses für Benutzerschnittstellen eingeführt. Er definiert die wiederverwendbare Wissensbasis für die Benutzerschnittstellengenerierung einer Anwendungsdomäne. Der „Domänenspezifische Baukasten“ setzt sich aus einem **Objektmetaphern-Katalog**, einem **Nutzungskontextmodell** und einer „Domain Theory“ zusammen.

Eclipse	Software-Plattform und Integrierte Entwicklungsumgebung (IDE). Weiterhin definiert Eclipse aber auch Layout- und Style-Vorgaben für den Aufbau von Benutzerschnittstellen.
Generisches Nutzungskontextmodell	Das generische Nutzungskontextmodell beschreibt, wie **Nutzungskontextmodelle** für unterschiedliche Anwendungsdomänen modelliert werden. Es spezifiziert die drei Komponenten Gerät, Nutzer, Umgebung der Nutzung und deren Beschreibung mit Hilfe von Kontextelementen.
Hardware platform Komponente	Die „Hardware platform“ Komponente erweitert UAProf durch die Beschreibungen von **Interaktions- und Ausgabemedien** für das **Nutzungsprofil**.
Hypertext	Hypertext sind Texte, die Informationen durch Verweisen auf externe Informationsobjekte (die wiederrum auf andere Informationsobjekte verweisen können) enthalten.
Input-Widget Komponente	Die Input-Widget Komponente erweitert UAProf durch die Beschreibungen von **Interaktionsobjekten** von Benutzerschnittstellen für das **Nutzungsprofil**.
Interaktionen	Interaktionen werden in dieser Arbeit basierend auf dem „Interaction Framework“ (vgl. [AB91]) definiert. Dabei wird zwischen **Benutzer**- und **Systeminteraktionen** unterschieden.
Interaktionsbaum	Ein Interaktionsbaum wird im Rahmen der Analyse eines BSF-AINs eingesetzt. Mit Hilfe des Interaktionsbaums können die Reihenfolgen aller Interaktionen eines BSF-AINs ermittelt werden.
Interaktionsbeschreibung	Die Interaktionsbeschreibung wird im Rahmen der Benutzerschnittstellengenerierung als Pendant zu der **Szenariobeschreibung** verwendet. Sie spezifiziert die Benutzerschnittstelle ohne Aussagen über das verwendete Endgerät, den Nutzer und die Umgebung der Nutzung zu treffen. Als Beschreibungebene wurde ein auf **Metaphern-basierter Ansatz** für die Beschreibung von Benutzerschnittstellen entwickelt. Spezifiziert wird die Interaktionsbeschreibung durch **Abstract Interaction Nets** (kurz AIN), eine auf Petri-Netzen basierende Modellierungsmethodik.
Interaktionsmedien	Physische Objekte zur Interaktion mit IT-Systemen wie beispielsweise Maus, Tastatur, Controller, Sprachsteuerung, usw. (vgl. **Ausgabemedien**).

Interaktionsmetaphern	Siehe **Metaphern-basierter Ansatz für die Interaktionsbeschreibung**.
Interaktionsobjekte	Elemente einer graphischen Benutzerschnittstelle wie beispielsweise Fenster, Buttons, Widgets, Listen, Menüs, usw.
Interaktionsraum	Siehe **Metaphern-basierter Ansatz für die Interaktionsbeschreibung**.
Intervallalgebra	Im Rahmen der Arbeit wird eine Intervallalgebra (in Anlehnung an Allen [All83]) verwendet, um Möglichkeiten für die Komposition von **Benutzerschnittstellenfragmenten** zu beschreiben. Für jede Relation der Intervallalgebra wird eine Methode spezifiziert, die beschreibt, wie die mit dieser Relation verknüften Benutzerschnittstellenfragmente komponiert werden.
Komplexe Transitionen	Komplexe Transitionen ermöglichen die Kapselung von komplexen Interaktionsabläufen in **Abstract Interaction Nets**, indem Unternetze, die ebenfalls **Abstract Interaction Nets** sind, für die Beschreibung dieser Interaktionen verwendet werden
Kontextorientierte Benutzerschnittstellen	Beschreibt die Vision der Arbeit, nämlich die Benutzerschnittstelle an die Bedürfnisse und Fähigkeiten des Benutzers, an das verwendete Endgerät und auch an die Umgebung der Nutzung anzupassen.
Layoutpattern	Ein Layoutpattern für Benutzerschnittstellen beschreibt Bereiche der Benutzerschnittstelle, in der Darstellungen realisiert sind, oder Bereiche, denen bestimmte Aufgaben zugeordnet wurden.
Layoutpattern Komponente	Die Layoutpattern Komponente erweitert UAProf durch die Beschreibungen von Strukturierungen der Oberfläche (vgl. **Layoutpattern**) der Benutzerschnittstelle.
Leitmetapher	Die Modellierung der **Interaktionsbeschreibung** mit einem **Metaphern-basierten Ansatz**, erfordert die Definition einer Leitmetapher, welche einen Kontext definiert, in dem die verwendeten **Objekt**- und **Interaktionsmetaphern** verwendet weden. Jedem **Interaktionsraum** wird eine Leitmetapher zugeordnet.
Metaphern-basierter Ansatz für die Interaktionsbeschreibung	Die Modellierung der **Interaktionsbeschreibung** folgt einem Metaphern-basierten Ansatz. Um Interaktionen eines Benutzers mit einer Benutzerschnittstelle unabhängig vom Endgerät, den Fähigkeiten des Nutzers und der Umgebung der Nutzung zu modellieren, werden Metaphern verwendet, die beschreiben, welche Interaktionen ein Nutzer durchführen kann.

	Ein **Interaktionsraum** spezifiziert dabei welche Objekte (**Objektmetaphern** genannt) für eine Interaktion zur Verfügung stehen. Mögliche Interaktionen mit den **Objektmetaphern** werden **Interaktionsmetaphern** genannt. Interaktionen können den **Interaktionsraum** verändern, indem neue **Objektmetaphern** erzeugt oder existierende entfernt werden.
Modellfragmente	Modellfragmente („domain model fragments“ in [FF91] genannt) sind im Rahmen des „**Compositional Modeling“** fundamentale Teile von Domäneneigenschaften wie Prozesse, Bauteile und Objekte und repräsentieren eine Wissensbasis, die durch den Kompositionsansatz zu neuem Wissen führen soll. Im Rahmen der vorliegenen Arbeit werden Modellfragmente durch **Benutzerschnittstellenfragmente** realisiert.
Multimodale Interaktionen	Mulitmodale Interaktionen beschreiben Eingaben, die sich gleichzeitig mehrerer Modalitäten wie Tastatur, Maus, aber auch Sprache, Gesten und Touch-Screens bedienen.
Nebenstellen	Nebenstellen sind Stellen im **Abstract Interaction Net**, die für eine Transition zugleich Eingangs- als auch Ausgangsstelle sind (vgl. Abschnitt 5.3.1).
Nutzungskontext	Ein Nutzungskontext beschreibt Anforderungen an die zu generierende Benutzerschnittstelle durch **Defaultprofile** und Kontextelemente.
Nutzungskontextmodell	Das Nutzungskontextmodell definiert **Defaultprofile** und Kontextelemente einer Anwendungsdomäne, mit denen **Nutzungskontexte** modelliert werden können. Das Nutzungskontextmodell ist Teil des **domänenspezifischen Baukastens** für die Generierung von Benutzerschnittstellen.
Nutzungskontextvektor	Jedem **Benutzerschnittstellenfragment** einer „Domain Theory“ wird ein Nutzungskontextvektor zugeordnet, der Informationen über Eigenschaften des Benutzerschnittstellenfragments beinhaltet. Beschrieben werden diese Eigenschaften über die Elemente des **Nutzungsprofils**. Der Nutzungskontextvektor wird verwendet, um zu ermitteln, wie gut ein **Benutzerschnittstellenfragment** den Anforderungen eines **Nutzungsprofils** gerecht wird.

Nutzungsprofil	Ein Nutzungsprofil wird aus einem **Nutzungskontext** generiert und beschreibt Anforderungen an die zu generierende Benutzerschnittstelle basierend auf einem CC/PP Profil. Das Nutzungsprofil setzt sich zusammen aus der **Input-Widget**, **Output Widget**, **Hardware platform** und der **Layoutpattern Komponente**.
Nutzungsprofilparameter	Nutzungsprofilparameter beschreiben Eigenschaften eines Kontextelementes (siehe **Nutzungskontextmodell**) und werden für die Generierung des **Nutzungsprofils** aus einem **Nutzungskontext** verwendet.
Objektmetaphern	Siehe **Metaphern-basierter Ansatz für die Interaktionsbeschreibung**.
Objektmetaphern-Katalog	Der Objektmetaphern-Katalog (kurz OM-Katalog) bildet die Wissensbasis für die Modellierung der Interaktionsbeschreibung, die alle Elemente des **Abstract Interaction Nets** (also alle **Objekt**- und **Interaktionsmetaphern**) umfasst, mit dem die **Interaktionsbeschreibung** definiert wird. Organisiert sind die **Interaktionsmetaphern** in **Abstrakten Interaktionen**, **Alternativen** und **Varianten**. Der OM-Katalog ist Teil des **domänenspezifischen Baukastens** für die Generierung von Benutzerschnittstellen.
Optionale Flüsse	Optionale Flüsse beschreiben Flussrelationen in **Dialognetzen**, die keine Bedingungen an das Schaltverhalten stellen, sich aber beim Schalten der Transition, falls es die Markierungsregeln erlauben, entsprechend verhalten.
Output-Widget Komponente	Die Output-Widget Komponente erweitert UAProf durch die Beschreibungen von Ausgaben (Bild, Text, Ton) von Benutzerschnittstellen für das **Nutzungsprofil.**
patientenorientierte telemedizinische Dienste	Patientenorientierte telemedizinische Dienste sind „telemedizinische Dienste“, die sich primär an Informations- und Unterstützungsbedürfnissen des Patienten orientieren. Zielsetzung ist es, im Behandlungsverlauf die **Compliance** des Patienten zu verbessern und ihn mittelfristig in die Lage zu versetzen, sich aktiv an seinem Gesundungsprozess zu beteiligen (Patient Empowerment).
Pervasive Computing	Pervasive Computing ist ein häufig verwendeter Begriff für Anwendungen, die über unterschiedliche (zumeist mobile) Endgeräte verfügbar gemacht werden.

Plastic user interface	Plastic user interfaces werden im Rahmen der Betrachtung von **adaptiven Benutzerschnittstellen** eingeführt. Sie werden in [TC99] durch das das Adaptionsziel definiert, die Benutzerschnittstelle an die physische Umgebung (Endgerät, Umfeld der Nutzung) unter Verwendung (Mittel zur Adaption) von Eingriffen in die Reihenfolge und Gestaltung der Teilaufgaben und Verwendung von unterschiedlichen Rendering-Techniken anzupassen.
Portlets	Ein Portlet kapselt die Benutzerschnittstelle für eine eigenständige Anwendung, die in ein bestehendes Web-Portal integriert wird, indem ein kleiner Ausschnitt der Oberfläche ausschließlich für das Portlet reserviert wird.
Präsentationsfragment	Präsentationsfragmente spezifizieren die Präsentation eines **Benutzerschnittstellenfragmentes**. Jedes Präsentationsfragment beschreibt dabei die Darstellung eines Interaktionsschrittes, bestehend aus der Darstellung der Inhalte und der **Interaktionsobjekte**, welche die Interaktionsmöglichkeiten eines Benutzers spezifizieren.
Query	Im Rahmen des „**Compositional Modeling“** definiert die Query das eigentliche Ziel des Generierungsprozesses (**Szenario Modell** genannt), nämlich auf welche Fragen das generierte Modell Antworten liefern soll. Im Rahmen der Benutzerschnittstellengenerierung wird die Query durch den **Nutzungskontext** realisiert.
Service-Orientierung	Beschreibt in dieser Arbeit Konzepte zur Modularisierung auf Basis von Software-Servcies.
Set-Top-Box	Eine Set-Top-Box ist ein Gerät, dass an einen Fernseher angeschlossen wird und als Computer fungiert, der über eine Fernbedienung (auch Maus und Tastatur sind möglich) gesteuert wird.
Smart Phone	Ein PDA mit dem zusätzlichen Funktionsumfang eines Mobiltelefons.
Source Domain	Source Domain beschreibt „das für einen Menschen Bekannte“ (Herkunftsbereich) im Rahmen der Conceptual Mataphor nach [LJ80].

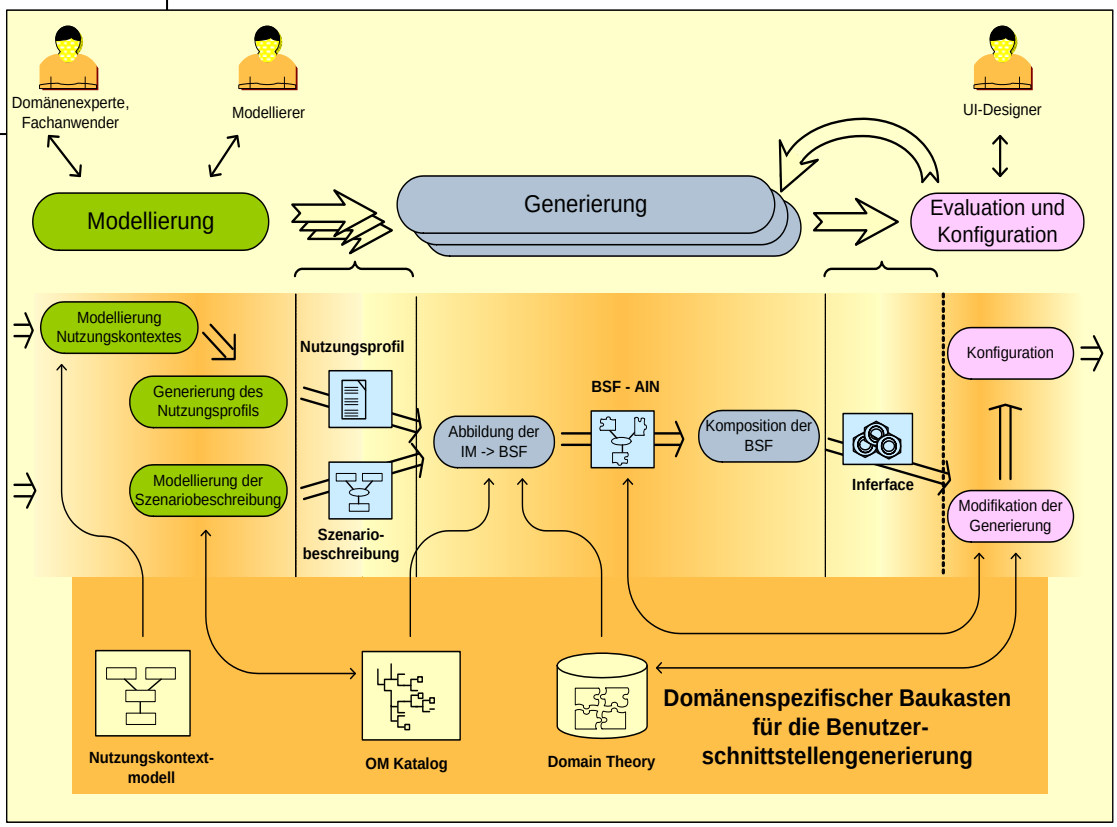
Systeminteraktionen	Beschreiben Transitionen in **Abstract Interaction Nets**, deren Schaltverhalten von externen Systemen (beispielsweise Datenbanken oder auch externe Dienste) abhängig ist. Unterschieden wird zwischen: „benutzerinitiierte Systeminteraktionen“, die durch den Benutzer ausgelöst werden, deren Ergebnis aber von externen Systemen abhängig ist; „externinitierte Systeminteraktionen“, die durch externe Systeme ausgelöst werden.
Szenariobeschreibung	Eine Szenariobeschreibung (in [FF91] „Scenario Description“ genannt) definiert im Rahmen des „**Compositional Modeling“** die Grundbausteine für die Generierung, indem sie einerseits die beteiligten Objekte benennt, andererseits aber auch festlegt, wie diese Objekte miteinander verknüpft sind und wie diese sich verhalten. Aus einer Szenariobeschreibung können dabei unterschiedliche Zielmodelle aufgebaut werden, die durch die **Query** bestimmt werden. Im Rahmen der Benutzerschnittstellengenerierung wird die Szenariobeschreibung durch die **Interaktionsbeschreibung** realisiert.
Scenario Model	Im Rahmen des „**Compositional Modeling“** beschreibt das Scenario Model das Ergebnis des Generierungsprozesses. Im Rahmen der vorliegenden Arbeit wird das Scenario Model durch die generierte Benutzerschnittstelle realisiert.
Target Domain	Target Domain beschreibt die „unbekannte bzw. abstrakte Umgebung“ (Zielbereich) im Rahmen der Conceptual Mataphor nach [LJ80].
Task-Modelle	Task-Modelle beschreiben die hierarchische Dekomposition von Tasks (Aufgaben) zur Erreichung eines bestimmten Zieles. Ziele können logische Aktivitäten sein, wie die Suche nach einer bestimmten Information, aber auch physische Aktivitäten, wie das Bestellen eines Taxis (vgl. [Pat01]).
Unified Reference Framework	Wurde in [GCT*03] definiert und wird in dieser Arbeit als Vorlage und gemeinsame Beschreibungsebene für unterschiedliche modellbasierte Systeme für die Benutzerschnittstellengenerierung verwendet.
Use-Cases	Use-Cases beschreiben in UML (vgl. [Oes05]) Anwendungsfälle aus Sicht des Anwenders, insbesondere beschreiben sie wie und mit welchem Ziel ein Anwender ein System nutzt.

Varianten (Interaktionen)	**Varianten** definieren im **Objektmetaphern-Katalog** unterschiedliche **Interaktionsmetaphern** die eine **Alternativen** realisieren. Beschrieben wird die **Interaktionsmetapher** entweder durch ein **Abstract Interaction Net** oder durch ein **Benutzerschnittstellenfragment**. Siehe **Abstrakte Interaktionen** und **Alternativen**.
Widgets	Komponenten einer graphischen Benutzerschnittstelle. Widgets können unter anderem Buttons, Fenster, Menüs, Listen etc. sein.

Abkürzungsverzeichnis

AAL	Ambient Assisted Living
AIN	Abstract Interaction Net
AS	Anwendungsschnittstelle
ASF	Anwendungsschnittstellenfragment
AUI	Abstract Interface
BMI	Body-Mass-Index
BSF	Benutzerschnittstellenfragmente
BSF-AIN	Ein Abstract Interaction Net für das Benutzerschnittstellenfragemente identifiziert wurden (siehe Glossar).
C&T	Concept and Task-Model
CAD	Computer Aided Design
CAD	Computer Aided Design
CC/PP	Composite Capabilities/Preference Profiles
CM-Prozess	„Compositional Modeling“-Prozess
CSP	Constraint Satisfaction Problem
CSS	Cascading Style Sheets
CTT	Concurrend Task Tree
CUI	Concrete User Interface.
DBMS	Database Management Systeme
DF	Dialogfragmente
DTD	Document Type Definition
eFA	Elektronische Fallakte
eGK	Elektronische Gesundheitskarte
EKG	Elektrokardiogramm
FUI	Final User Interface

GOMS	Goals, Operators, Methods and Selection Rules
GUI	Graphical User Interfaces
HBA	Heilberufeausweis
HCI	Human Computer Interaction
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IM	Interaktionsmetapher
J2EE	Java Platform, Enterprise Edition
MCE	Microsoft Media Center Edition
MDA	Model Driven Architecture
MFC	Microsoft Foundation Classes
MVC	Model View Control
OASIS	Organization for the Advancement of Structured Information Standards
OM	Objektmetapher
OM-Katalog	Objektmetaphern-Katalog
PAC	Presentation, Abstraction, Control
PDA	Personal Digital Assistant
PF	Präsentationsfragmente
RDF	Resource Description Framework
RIA	Rich Internet Application
UAProf	User Agent Profile
UI	User Interface
UIML	User Interface Markup Language
UIMS	User Interface Management Systeme
UML	Unified Modeling Language
UsiXML	User Interface Extensible Markup Language



XML Extensible Markup Language

XUL XML User Interface Language